

DNG SDK 1.3

Generated by Doxygen 1.5.9

Sat Jun 20 17:35:20 2009

## Contents

<b>1</b>	<b>Adobe Digital Negative SDK 1.3</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Command line validation: dng_validate . . . . .	2
1.3	Starting points . . . . .	2
1.4	Related documentation . . . . .	2
<b>2</b>	<b>doc_dng_validate</b>	<b>3</b>
<b>3</b>	<b>Class Index</b>	<b>4</b>
3.1	Class Hierarchy . . . . .	4
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>11</b>
5.1	File List . . . . .	11
<b>6</b>	<b>Class Documentation</b>	<b>14</b>
6.1	AutoPtr< T > Class Template Reference . . . . .	14
6.1.1	Detailed Description . . . . .	15
6.1.2	Constructor & Destructor Documentation . . . . .	15
6.2	dng_1d_concatenate Class Reference . . . . .	16
6.2.1	Detailed Description . . . . .	16
6.2.2	Constructor & Destructor Documentation . . . . .	16
6.2.3	Member Function Documentation . . . . .	17
6.3	dng_1d_function Class Reference . . . . .	18
6.3.1	Detailed Description . . . . .	18
6.3.2	Member Function Documentation . . . . .	18
6.4	dng_1d_identity Class Reference . . . . .	19
6.4.1	Detailed Description . . . . .	20
6.5	dng_1d_inverse Class Reference . . . . .	20
6.5.1	Detailed Description . . . . .	21

---

6.5.2	Member Function Documentation	21
6.6	dng_1d_table Class Reference	22
6.6.1	Detailed Description	22
6.6.2	Member Function Documentation	23
6.7	dng_abort_sniffer Class Reference	23
6.7.1	Detailed Description	24
6.7.2	Member Function Documentation	24
6.8	dng_area_task Class Reference	26
6.8.1	Detailed Description	27
6.8.2	Member Function Documentation	27
6.9	dng_camera_profile Class Reference	32
6.9.1	Detailed Description	35
6.9.2	Member Function Documentation	35
6.10	dng_color_space Class Reference	40
6.10.1	Detailed Description	42
6.10.2	Member Function Documentation	42
6.11	dng_color_spec Class Reference	42
6.11.1	Detailed Description	43
6.11.2	Constructor & Destructor Documentation	43
6.11.3	Member Function Documentation	43
6.12	dng_const_tile_buffer Class Reference	45
6.12.1	Detailed Description	45
6.12.2	Constructor & Destructor Documentation	45
6.13	dng_date_time Class Reference	46
6.13.1	Detailed Description	46
6.13.2	Constructor & Destructor Documentation	47
6.13.3	Member Function Documentation	47
6.14	dng_date_time_info Class Reference	48
6.14.1	Detailed Description	49
6.15	dng_date_time_storage_info Class Reference	49
6.15.1	Detailed Description	49

---

6.15.2	Member Function Documentation	49
6.16	dng_dirty_tile_buffer Class Reference	50
6.16.1	Detailed Description	51
6.16.2	Constructor & Destructor Documentation	51
6.17	dng_exception Class Reference	51
6.17.1	Detailed Description	51
6.17.2	Constructor & Destructor Documentation	52
6.17.3	Member Function Documentation	52
6.18	dng_exif Class Reference	52
6.18.1	Detailed Description	56
6.19	dng_file_stream Class Reference	56
6.19.1	Detailed Description	56
6.19.2	Constructor & Destructor Documentation	57
6.20	dng_filter_task Class Reference	57
6.20.1	Detailed Description	58
6.20.2	Constructor & Destructor Documentation	58
6.20.3	Member Function Documentation	58
6.21	dng_fingerprint Class Reference	61
6.21.1	Detailed Description	61
6.22	dng_function_exposure_ramp Class Reference	62
6.22.1	Detailed Description	62
6.22.2	Member Function Documentation	62
6.23	dng_function_exposure_tone Class Reference	63
6.23.1	Detailed Description	63
6.24	dng_function_gamma_encode Class Reference	64
6.24.1	Detailed Description	64
6.24.2	Member Function Documentation	64
6.25	dng_function_GammaEncode_1_8 Class Reference	65
6.25.1	Detailed Description	65
6.25.2	Member Function Documentation	65
6.26	dng_function_GammaEncode_2_2 Class Reference	66

---

6.26.1 Detailed Description . . . . .	66
6.26.2 Member Function Documentation . . . . .	67
6.27 dng_function_GammaEncode_sRGB Class Reference . . . . .	68
6.27.1 Detailed Description . . . . .	68
6.27.2 Member Function Documentation . . . . .	68
6.28 dng_host Class Reference . . . . .	69
6.28.1 Detailed Description . . . . .	71
6.28.2 Constructor & Destructor Documentation . . . . .	71
6.28.3 Member Function Documentation . . . . .	72
6.29 dng_ifd Class Reference . . . . .	77
6.29.1 Detailed Description . . . . .	80
6.30 dng_image Class Reference . . . . .	81
6.30.1 Detailed Description . . . . .	83
6.30.2 Member Enumeration Documentation . . . . .	83
6.30.3 Member Function Documentation . . . . .	83
6.31 dng_image_writer Class Reference . . . . .	87
6.31.1 Detailed Description . . . . .	88
6.31.2 Member Function Documentation . . . . .	88
6.32 dng_info Class Reference . . . . .	91
6.32.1 Detailed Description . . . . .	92
6.32.2 Member Function Documentation . . . . .	92
6.33 dng_iptc Class Reference . . . . .	93
6.33.1 Detailed Description . . . . .	95
6.33.2 Member Function Documentation . . . . .	95
6.34 dng_linearization_info Class Reference . . . . .	96
6.34.1 Detailed Description . . . . .	98
6.34.2 Member Function Documentation . . . . .	98
6.34.3 Member Data Documentation . . . . .	99
6.35 dng_memory_allocator Class Reference . . . . .	100
6.35.1 Detailed Description . . . . .	100
6.35.2 Member Function Documentation . . . . .	100

---

6.36	dng_memory_block Class Reference . . . . .	101
6.36.1	Detailed Description . . . . .	102
6.36.2	Member Function Documentation . . . . .	102
6.37	dng_memory_data Class Reference . . . . .	107
6.37.1	Detailed Description . . . . .	108
6.37.2	Constructor & Destructor Documentation . . . . .	108
6.37.3	Member Function Documentation . . . . .	109
6.38	dng_memory_stream Class Reference . . . . .	115
6.38.1	Detailed Description . . . . .	115
6.38.2	Constructor & Destructor Documentation . . . . .	116
6.38.3	Member Function Documentation . . . . .	116
6.39	dng_mosaic_info Class Reference . . . . .	116
6.39.1	Detailed Description . . . . .	118
6.39.2	Member Function Documentation . . . . .	118
6.39.3	Member Data Documentation . . . . .	121
6.40	dng_negative Class Reference . . . . .	122
6.40.1	Detailed Description . . . . .	133
6.40.2	Member Function Documentation . . . . .	133
6.41	dng_noise_function Class Reference . . . . .	135
6.41.1	Detailed Description . . . . .	135
6.41.2	Member Function Documentation . . . . .	135
6.42	dng_noise_profile Class Reference . . . . .	136
6.42.1	Detailed Description . . . . .	136
6.43	dng_opcode_FixVignetteRadial Class Reference . . . . .	137
6.43.1	Detailed Description . . . . .	138
6.44	dng_opcode_WarpFisheye Class Reference . . . . .	138
6.44.1	Detailed Description . . . . .	139
6.45	dng_opcode_WarpRectilinear Class Reference . . . . .	139
6.45.1	Detailed Description . . . . .	140
6.46	dng_pixel_buffer Class Reference . . . . .	140
6.46.1	Detailed Description . . . . .	142

---

6.46.2	Member Function Documentation	142
6.47	dng_render Class Reference	156
6.47.1	Detailed Description	156
6.47.2	Constructor & Destructor Documentation	157
6.47.3	Member Function Documentation	157
6.48	dng_simple_image Class Reference	161
6.48.1	Detailed Description	162
6.49	dng_sniffer_task Class Reference	162
6.49.1	Detailed Description	162
6.49.2	Constructor & Destructor Documentation	163
6.49.3	Member Function Documentation	163
6.50	dng_space_AdobeRGB Class Reference	164
6.50.1	Detailed Description	164
6.51	dng_space_ColorMatch Class Reference	165
6.51.1	Detailed Description	165
6.52	dng_space_GrayGamma18 Class Reference	165
6.52.1	Detailed Description	166
6.53	dng_space_GrayGamma22 Class Reference	166
6.53.1	Detailed Description	167
6.54	dng_space_ProPhoto Class Reference	167
6.54.1	Detailed Description	168
6.55	dng_space_sRGB Class Reference	168
6.55.1	Detailed Description	168
6.56	dng_stream Class Reference	169
6.56.1	Detailed Description	171
6.56.2	Constructor & Destructor Documentation	171
6.56.3	Member Function Documentation	171
6.57	dng_tile_buffer Class Reference	188
6.57.1	Detailed Description	188
6.57.2	Constructor & Destructor Documentation	188
6.58	dng_time_zone Class Reference	189

---

6.58.1 Detailed Description . . . . .	189
6.59 dng_tone_curve_acr3_default Class Reference . . . . .	190
6.59.1 Detailed Description . . . . .	190
6.60 dng_vignette_radial_params Class Reference . . . . .	190
6.60.1 Detailed Description . . . . .	191
6.61 dng_warp_params Class Reference . . . . .	191
6.61.1 Detailed Description . . . . .	192
6.62 dng_warp_params_fisheye Class Reference . . . . .	192
6.62.1 Detailed Description . . . . .	193
6.63 dng_warp_params_rectilinear Class Reference . . . . .	193
6.63.1 Detailed Description . . . . .	194
<b>7 File Documentation</b>	<b>194</b>
7.1 dng_1d_function.h File Reference . . . . .	194
7.1.1 Detailed Description . . . . .	195
7.2 dng_1d_table.h File Reference . . . . .	195
7.2.1 Detailed Description . . . . .	195
7.3 dng_abort_sniffer.h File Reference . . . . .	195
7.3.1 Detailed Description . . . . .	196
7.4 dng_area_task.h File Reference . . . . .	196
7.4.1 Detailed Description . . . . .	196
7.5 dng_assertions.h File Reference . . . . .	196
7.5.1 Detailed Description . . . . .	196
7.5.2 Define Documentation . . . . .	196
7.6 dng_auto_ptr.h File Reference . . . . .	198
7.6.1 Detailed Description . . . . .	198
7.7 dng_bottlenecks.h File Reference . . . . .	198
7.7.1 Detailed Description . . . . .	204
7.8 dng_camera_profile.h File Reference . . . . .	204
7.8.1 Detailed Description . . . . .	205
7.9 dng_color_space.h File Reference . . . . .	205

---

7.9.1 Detailed Description . . . . .	206
7.10 dng_color_spec.h File Reference . . . . .	206
7.10.1 Detailed Description . . . . .	206
7.10.2 Function Documentation . . . . .	206
7.11 dng_date_time.h File Reference . . . . .	207
7.11.1 Detailed Description . . . . .	207
7.11.2 Enumeration Type Documentation . . . . .	208
7.11.3 Function Documentation . . . . .	208
7.12 dng_errors.h File Reference . . . . .	209
7.12.1 Detailed Description . . . . .	209
7.13 dng_exceptions.h File Reference . . . . .	209
7.13.1 Detailed Description . . . . .	211
7.14 dng_exif.h File Reference . . . . .	211
7.14.1 Detailed Description . . . . .	211
7.15 dng_fast_module.h File Reference . . . . .	212
7.15.1 Detailed Description . . . . .	212
7.16 dng_file_stream.h File Reference . . . . .	212
7.16.1 Detailed Description . . . . .	212
7.17 dng_filter_task.h File Reference . . . . .	212
7.17.1 Detailed Description . . . . .	212
7.18 dng_fingerprint.h File Reference . . . . .	212
7.18.1 Detailed Description . . . . .	213
7.19 dng_flags.h File Reference . . . . .	213
7.19.1 Detailed Description . . . . .	213
7.20 dng_globals.h File Reference . . . . .	213
7.20.1 Detailed Description . . . . .	213
7.21 dng_host.h File Reference . . . . .	213
7.21.1 Detailed Description . . . . .	214
7.22 dng_ifd.h File Reference . . . . .	214
7.22.1 Detailed Description . . . . .	214
7.23 dng_image.h File Reference . . . . .	214

---

7.23.1 Detailed Description . . . . .	215
7.24 dng_image_writer.h File Reference . . . . .	215
7.24.1 Detailed Description . . . . .	216
7.25 dng_info.h File Reference . . . . .	216
7.25.1 Detailed Description . . . . .	216
7.26 dng_ipvc.h File Reference . . . . .	216
7.26.1 Detailed Description . . . . .	216
7.27 dng_linearization_info.h File Reference . . . . .	216
7.27.1 Detailed Description . . . . .	216
7.28 dng_lossless_jpeg.h File Reference . . . . .	217
7.28.1 Detailed Description . . . . .	217
7.29 dng_matrix.h File Reference . . . . .	217
7.29.1 Detailed Description . . . . .	218
7.30 dng_memory_stream.h File Reference . . . . .	218
7.30.1 Detailed Description . . . . .	218
7.31 dng_mosaic_info.h File Reference . . . . .	218
7.31.1 Detailed Description . . . . .	218
7.32 dng_negative.h File Reference . . . . .	218
7.32.1 Detailed Description . . . . .	219
7.33 dng_pixel_buffer.h File Reference . . . . .	219
7.33.1 Detailed Description . . . . .	219
7.34 dng_read_image.h File Reference . . . . .	220
7.34.1 Detailed Description . . . . .	220
7.35 dng_render.h File Reference . . . . .	220
7.35.1 Detailed Description . . . . .	220
7.36 dng_sdk_limits.h File Reference . . . . .	221
7.36.1 Detailed Description . . . . .	221
7.36.2 Variable Documentation . . . . .	221

# 1 Adobe Digital Negative SDK 1.3

## 1.1 Introduction

Digital Negative (DNG) is a non-proprietary file format for camera raw image data and metadata. A wide variety of cameras and sensor types are supported by DNG, using the same documented file layout.

This SDK provides support for reading and writing DNG files as well as support for converting DNG data into a displayable or processible image. This SDK is intended to serve as a starting point for adding DNG support to existing applications that use and manipulate images.

## 1.2 Command line validation: `dng_validate`

A good place to start investigating the DNG SDK is the `dng_validate` command line tool, which can read, validate and convert an existing DNG file. The `dng_validate.cpp` file demonstrates a number of common uses of the SDK. Documentation for the tool can be found [here](#).

## 1.3 Starting points

- [dng\\_host](#) Used to customize memory allocation, to communicate progress updates and test for cancellation.
- [dng\\_negative](#) Main container for metadata and image data in a DNG file.
- [dng\\_image](#) Class used to hold and manipulate image data.
- [dng\\_renderer](#) Class used to convert DNG RAW data to displayable image data.
- [dng\\_image\\_writer](#) Class used to write DNG files.

## 1.4 Related documentation

- The Adobe Digital Negative specification: [http://www.adobe.com/products/dng/pdfs/dng\\_spec.pdf](http://www.adobe.com/products/dng/pdfs/dng_spec.pdf)
- TIFF 6 specification: <http://partners.adobe.com/public/developer/tiff/index.html>
- TIFF/EP specification: <http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail>
- EXIF specification: [http://www.jeita.or.jp/english/standard/html/1\\_4.htm](http://www.jeita.or.jp/english/standard/html/1_4.htm)
- IPTC specification: <http://www.iptc.org/IPTC7901/>

## 2 doc\_dng\_validate

dng\_validate Version 1.3 20-Jun-09

“dng\_validate” is a command-line tool that parses the tag structure of DNG (and other TIFF-EP based format) files, and reports any deviations from the DNG specification that it finds.

The usage syntax is:

```
dng_validate [-v] [-d <number>] [-f] [-b4] [-s < CFA index >] [-q <target-binned-width>] [-cs1|-cs2|-cs3|-cs4|-cs5|-cs6] [-16] { [-1 <stage1-out-filename> ] [-2 <stage2-out-filename> ] [-3 <stage3-out-filename> ] [-tif <TIFF-out-filename>] [-dng <DNG-out-file>] {<list of files>}*
```

Any deviations from the DNG specification are written to the standard error stream.

The “-v” option turns on “verbose” mode, which writes the parsed tag structure to the standard output stream. Any tags that are not parsed by this tool are preceded by an asterisk.

The “-d <number>” option both implies verbose mode, and also specifies the maximum number of lines of data displayed per tag.

The “-f” option switches dng\_validate to using floating-point math where possible, instead of the default 16-bit integer.

The “-b4” option causes the demosaic algorithm to produce a four-channel output rather than a three-channel one. (The input DNG must be a three-channel Bayer pattern image.) This option is only useful when used with the -3 switch. The extra channel is the result of doing two interpolations of the Bayer green channel such that the greens on the same row as the reds produces one channel and the greens on the same row as the blues produce another channel. The second green channel will be the highest numbered channel in the output. This option is used to gauge the difference between greens in each row to decide whether the DNG BayerGreenSplit tag should be used for a given source of image data (e.g. camera).

The “-s <CFA index>” option chooses which set of color filter arrays to use when there are multiple ones for an input image. Each CFA array is a separate channel in the DNG input. This applies to the Fuji SR cameras for example, where the first channel is from the S-sensing elements and the second channel is from the R-sensing elements. The S elements are more sensitive and the R elements are less so with the goal of using both to increase the dynamic range the sensor can capture in a single image. By default dng\_validate generates an image from only the S-sensors. By using “-s 1” the R-sensing elements’ data can be used to construct the output image. (This index is 0-based. The default is 0.)

The “-q <target-binned-size>” option enables binning during the demosaic process. This is useful for creating previews or thumbnails. The binning factor is determined from the target-binned-size, which is the size in pixels of the larger dimension of the

image that is desired. An integer binning factor will be computed to produce an image of that size or larger. For example, if the input image is 3008 x 2000 pixels and the target-binned-width is 700, factor of 4 binning will be used and the result output image (after demosaicing) will be 752 x 500 pixels.

The “-cs1” option generates the output image in sRGB color space.

The “-cs2” option generates the output image in AdobeRGB color space.

The “-cs3” option generates the output image in ProPhotoRGB color space.

The “-cs4” option generates the output image in ColorMatch color space.

The “-cs5” option generates the output image in grayscale gamma 1.8 color space.

The “-cs6” option generates the output image in a grayscale gamma 2.2 color space.

The “-16” option causes dng\_validate to output 16-bit-per-component images rather than the default 8-bit.

The “-1” option causes the unprocessed raw image data to be written to the named output file. This applies only to the next input file after the switch.

The “-2” option causes the image data after linearization and black/white level mapping to be written to the named output file. This applies only to the next input file after the switch.

The “-3” option causes the image data after demosaic processing, but prior to color space conversion, noise reduction, sharpening, etc., to be written to the named output file. This applies only to the next input file after the switch.

The “-tif” option causes the final rendered image to be written as TIFF to the named output file. This applies only to the next input file after the switch.

The “-dng” option causes the parsed DNG data to be reserialized and written to the named output file. This mostly serves to provide an example code path for the process of writing a DNG file, as the output may not differ significantly from the input DNG. (Parameters, such as whether the data is compressed or not, may vary between the input and output DNG files.) This applies only to the next input file after the switch.

## 3 Class Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<b>AutoPtr&lt; T &gt;</b>	<b>14</b>
<b>dng_1d_function</b>	<b>18</b>
<b>dng_1d_concatenate</b>	<b>16</b>

---

<b>dng_1d_identity</b>	<b>19</b>
<b>dng_1d_inverse</b>	<b>20</b>
<b>dng_function_exposure_ramp</b>	<b>62</b>
<b>dng_function_exposure_tone</b>	<b>63</b>
<b>dng_function_gamma_encode</b>	<b>64</b>
<b>dng_function_GammaEncode_1_8</b>	<b>65</b>
<b>dng_function_GammaEncode_2_2</b>	<b>66</b>
<b>dng_function_GammaEncode_sRGB</b>	<b>68</b>
<b>dng_noise_function</b>	<b>135</b>
<b>dng_tone_curve_acr3_default</b>	<b>190</b>
<b>dng_1d_table</b>	<b>22</b>
<b>dng_abort_sniffer</b>	<b>23</b>
<b>dng_area_task</b>	<b>26</b>
<b>dng_filter_task</b>	<b>57</b>
<b>dng_camera_profile</b>	<b>32</b>
<b>dng_color_space</b>	<b>40</b>
<b>dng_space_AdobeRGB</b>	<b>164</b>
<b>dng_space_ColorMatch</b>	<b>165</b>
<b>dng_space_GrayGamma18</b>	<b>165</b>
<b>dng_space_GrayGamma22</b>	<b>166</b>
<b>dng_space_ProPhoto</b>	<b>167</b>
<b>dng_space_sRGB</b>	<b>168</b>
<b>dng_color_spec</b>	<b>42</b>
<b>dng_date_time</b>	<b>46</b>
<b>dng_date_time_info</b>	<b>48</b>

---

<b>dng_date_time_storage_info</b>	<b>49</b>
<b>dng_exception</b>	<b>51</b>
<b>dng_exif</b>	<b>52</b>
<b>dng_fingerprint</b>	<b>61</b>
<b>dng_host</b>	<b>69</b>
<b>dng_ifd</b>	<b>77</b>
<b>dng_image</b>	<b>81</b>
<b>dng_simple_image</b>	<b>161</b>
<b>dng_image_writer</b>	<b>87</b>
<b>dng_info</b>	<b>91</b>
<b>dng_iptc</b>	<b>93</b>
<b>dng_linearization_info</b>	<b>96</b>
<b>dng_memory_allocator</b>	<b>100</b>
<b>dng_memory_block</b>	<b>101</b>
<b>dng_memory_data</b>	<b>107</b>
<b>dng_mosaic_info</b>	<b>116</b>
<b>dng_negative</b>	<b>122</b>
<b>dng_noise_profile</b>	<b>136</b>
<b>dng_opcode</b>	
<b>dng_inplace_opcode</b>	
<b>dng_opcode_FixVignetteRadial</b>	<b>137</b>
<b>dng_opcode_WarpFisheye</b>	<b>138</b>
<b>dng_opcode_WarpRectilinear</b>	<b>139</b>
<b>dng_pixel_buffer</b>	<b>140</b>
<b>dng_tile_buffer</b>	<b>188</b>
<b>dng_const_tile_buffer</b>	<b>45</b>

---

<b>dng_dirty_tile_buffer</b>	<b>50</b>
<b>dng_render</b>	<b>156</b>
<b>dng_sniffer_task</b>	<b>162</b>
<b>dng_stream</b>	<b>169</b>
<b>dng_file_stream</b>	<b>56</b>
<b>dng_memory_stream</b>	<b>115</b>
<b>dng_time_zone</b>	<b>189</b>
<b>dng_vignette_radial_params</b>	<b>190</b>
<b>dng_warp_params</b>	<b>191</b>
<b>dng_warp_params_fisheye</b>	<b>192</b>
<b>dng_warp_params_rectilinear</b>	<b>193</b>

## 4 Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<b><a href="#">AutoPtr&lt; T &gt;</a></b> (A class intended to be used in stack scope to hold a pointer from new. The held pointer will be deleted automatically if the scope is left without calling Release on the <a href="#">AutoPtr</a> first )	<b>14</b>
<b><a href="#">dng_1d_concatenate</a></b> (A <a href="#">dng_1d_function</a> that represents the composition (curry) of two other <a href="#">dng_1d_functions</a> )	<b>16</b>
<b><a href="#">dng_1d_function</a></b> (A 1D floating-point function )	<b>18</b>
<b><a href="#">dng_1d_identity</a></b> (An identity (x -> y such that x == y for all x) mapping function )	<b>19</b>
<b><a href="#">dng_1d_inverse</a></b> (A <a href="#">dng_1d_function</a> that represents the inverse of another <a href="#">dng_1d_function</a> )	<b>20</b>
<b><a href="#">dng_1d_table</a></b> (A 1D floating-point lookup table using linear interpolation )	<b>22</b>

---

<a href="#">dng_abort_sniffer</a> (Class for signaling user cancellation and receiving progress updates )	23
<a href="#">dng_area_task</a> (Abstract class for rectangular processing operations with support for partitioning across multiple processing resources and observing memory constraints )	26
<a href="#">dng_camera_profile</a> (Container for DNG camera color profile and calibration data )	32
<a href="#">dng_color_space</a> (An abstract color space )	40
<a href="#">dng_color_spec</a>	42
<a href="#">dng_const_tile_buffer</a> (Class to get resource acquisition is instantiation behavior for constant (read-only) tile buffers )	45
<a href="#">dng_date_time</a> (Class for holding a date/time and converting to and from relevant date/time formats )	46
<a href="#">dng_date_time_info</a> (Class for holding complete data/time/zone information )	48
<a href="#">dng_date_time_storage_info</a> (Store file offset from which date was read )	49
<a href="#">dng_dirty_tile_buffer</a> (Class to get resource acquisition is instantiation behavior for dirty (writable) tile buffers )	50
<a href="#">dng_exception</a> (All exceptions thrown by the DNG SDK use this exception class )	51
<a href="#">dng_exif</a> (Container class for parsing and holding EXIF tags )	52
<a href="#">dng_file_stream</a> (A stream to/from a disk file. See <a href="#">dng_stream</a> for read/write interface )	56
<a href="#">dng_filter_task</a> (Represents a task which filters an area of a source <a href="#">dng_image</a> to an area of a destination <a href="#">dng_image</a> )	57
<a href="#">dng_fingerprint</a> (Container fingerprint (MD5 only at present) )	61
<a href="#">dng_function_exposure_ramp</a> (Curve for pre-exposure-compensation adjustment based on noise floor, shadows, and highlight level )	62
<a href="#">dng_function_exposure_tone</a> (Exposure compensation curve for a given compensation amount in stops using quadric for roll-off )	63
<a href="#">dng_function_gamma_encode</a> (Encoding gamma curve for a given color	

---

space )	64
<a href="#">dng_function_GammaEncode_1_8</a> (A <a href="#">dng_1d_function</a> for gamma encoding with 1.8 gamma )	65
<a href="#">dng_function_GammaEncode_2_2</a> (A <a href="#">dng_1d_function</a> for gamma encoding with 2.2 gamma )	66
<a href="#">dng_function_GammaEncode_sRGB</a> (A <a href="#">dng_1d_function</a> for gamma encoding in sRGB color space )	68
<a href="#">dng_host</a> (The main class for communication between the application and the DNG SDK. Used to customize memory allocation and other behaviors )	69
<a href="#">dng_ifd</a> (Container for a single image file directory of a digital negative )	77
<a href="#">dng_image</a> (Base class for holding image data in DNG SDK. See <a href="#">dng_simple_image</a> for derived class most often used in DNG SDK )	81
<a href="#">dng_image_writer</a> (Support for writing <a href="#">dng_image</a> or <a href="#">dng_negative</a> instances to a <a href="#">dng_stream</a> in TIFF or DNG format )	87
<a href="#">dng_info</a> (Top-level structure of DNG file with access to metadata )	91
<a href="#">dng_iptc</a> (Class for reading and holding IPTC metadata associated with a DNG file )	93
<a href="#">dng_linearization_info</a> (Class for managing data values related to DNG linearization )	96
<a href="#">dng_memory_allocator</a> (Interface for <a href="#">dng_memory_block</a> allocator )	100
<a href="#">dng_memory_block</a> (Class to provide resource acquisition is instantiation discipline for image buffers and other larger memory allocations )	101
<a href="#">dng_memory_data</a> (Class to provide resource acquisition is instantiation discipline for small memory allocations )	107
<a href="#">dng_memory_stream</a> (A <a href="#">dng_stream</a> which can be read from or written to memory )	115
<a href="#">dng_mosaic_info</a> (Support for describing color filter array patterns and manipulating mosaic sample data )	116
<a href="#">dng_negative</a> (Main class for holding DNG image data and associated metadata )	122

---

<a href="#">dng_noise_function</a> (Noise model for photon and sensor read noise, assuming that they are independent random variables and spatially invariant )	135
<a href="#">dng_noise_profile</a> (Noise profile for a negative )	136
<a href="#">dng_opcode_FixVignetteRadial</a> (Radially-symmetric lens vignette correction opcode )	137
<a href="#">dng_opcode_WarpFisheye</a> (Warp opcode for fisheye camera model )	138
<a href="#">dng_opcode_WarpRectilinear</a> (Warp opcode for pinhole perspective (rectilinear) camera model )	139
<a href="#">dng_pixel_buffer</a> (Holds a buffer of pixel data with "pixel geometry" metadata )	140
<a href="#">dng_render</a> (Class used to render digital negative to displayable image )	156
<a href="#">dng_simple_image</a> (Dng_image derived class with simple Trim and Rotate functionality )	161
<a href="#">dng_sniffer_task</a> (Class to establish scope of a named subtask in DNG processing )	162
<a href="#">dng_space_AdobeRGB</a> (Singleton class for AdobeRGB color space )	164
<a href="#">dng_space_ColorMatch</a> (Singleton class for ColorMatch color space )	165
<a href="#">dng_space_GrayGamma18</a> (Singleton class for gamma 1.8 grayscale color space )	165
<a href="#">dng_space_GrayGamma22</a> (Singleton class for gamma 2.2 grayscale color space )	166
<a href="#">dng_space_ProPhoto</a> (Singleton class for ProPhoto RGB color space )	167
<a href="#">dng_space_sRGB</a> (Singleton class for sRGB color space )	168
<a href="#">dng_stream</a>	169
<a href="#">dng_tile_buffer</a> (Class to get resource acquisition is instantiation behavior for tile buffers. Can be dirty or constant tile access )	188
<a href="#">dng_time_zone</a> (Class for holding a time zone )	189
<a href="#">dng_tone_curve_acr3_default</a> (Default ACR3 tone curve )	190

---

<a href="#">dng_vignette_radial_params</a> (Radially-symmetric vignette (peripheral illumination falloff) correction parameters )	190
<a href="#">dng_warp_params</a> (Abstract base class holding common warp opcode parameters (e.g., number of planes, optical center) and common warp routines )	191
<a href="#">dng_warp_params_fisheye</a> (Warp parameters for fisheye camera model (radial component only). Note the restrictions described below )	192
<a href="#">dng_warp_params_rectilinear</a> (Warp parameters for pinhole perspective rectilinear (not fisheye) camera model. Supports radial and tangential (decentering) distortion correction parameters )	193

## 5 File Index

### 5.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">dng_1d_function.h</a>	194
<a href="#">dng_1d_table.h</a>	195
<a href="#">dng_abort_sniffer.h</a>	195
<a href="#">dng_area_task.h</a>	196
<a href="#">dng_assertions.h</a>	196
<a href="#">dng_auto_ptr.h</a>	198
<a href="#">dng_bad_pixels.h</a>	??
<a href="#">dng_bottlenecks.h</a>	198
<a href="#">dng_camera_profile.h</a>	204
<a href="#">dng_classes.h</a>	??
<a href="#">dng_color_space.h</a>	205
<a href="#">dng_color_spec.h</a>	206
<a href="#">dng_date_time.h</a>	207

---

<a href="#">dng_errors.h</a>	209
<a href="#">dng_exceptions.h</a>	209
<a href="#">dng_exif.h</a>	211
<a href="#">dng_fast_module.h</a>	212
<a href="#">dng_file_stream.h</a>	212
<a href="#">dng_filter_task.h</a>	212
<a href="#">dng_fingerprint.h</a>	212
<a href="#">dng_flags.h</a>	213
<a href="#">dng_gain_map.h</a>	??
<a href="#">dng_globals.h</a>	213
<a href="#">dng_host.h</a>	213
<a href="#">dng_hue_sat_map.h</a>	??
<a href="#">dng_ifd.h</a>	214
<a href="#">dng_image.h</a>	214
<a href="#">dng_image_writer.h</a>	215
<a href="#">dng_info.h</a>	216
<a href="#">dng_iptc.h</a>	216
<a href="#">dng_lens_correction.h</a>	??
<a href="#">dng_linearization_info.h</a>	216
<a href="#">dng_lossless_jpeg.h</a>	217
<a href="#">dng_matrix.h</a>	217
<a href="#">dng_memory.h</a>	??
<a href="#">dng_memory_stream.h</a>	218
<a href="#">dng_misc_opcodes.h</a>	??
<a href="#">dng_mosaic_info.h</a>	218

---

dng_mutex.h	??
<a href="#">dng_negative.h</a>	<a href="#">218</a>
dng_opcode_list.h	??
dng_opcodes.h	??
dng_orientation.h	??
dng_parse_utils.h	??
<a href="#">dng_pixel_buffer.h</a>	<a href="#">219</a>
dng_point.h	??
dng_preview.h	??
dng_pthread.h	??
dng_rational.h	??
<a href="#">dng_read_image.h</a>	<a href="#">220</a>
dng_rect.h	??
dng_reference.h	??
<a href="#">dng_render.h</a>	<a href="#">220</a>
dng_resample.h	??
<a href="#">dng_sdk_limits.h</a>	<a href="#">221</a>
dng_shared.h	??
dng_simple_image.h	??
dng_spline.h	??
dng_stream.h	??
dng_string.h	??
dng_string_list.h	??
dng_tag_codes.h	??
dng_tag_types.h	??

---

<b>dng_tag_values.h</b>	??
<b>dng_temperature.h</b>	??
<b>dng_tile_iterator.h</b>	??
<b>dng_tone_curve.h</b>	??
<b>dng_types.h</b>	??
<b>dng_utils.h</b>	??
<b>dng_xmp.h</b>	??
<b>dng_xmp_sdk.h</b>	??
<b>dng_xy_coord.h</b>	??

## 6 Class Documentation

### 6.1 `AutoPtr< T >` Class Template Reference

A class intended to be used in stack scope to hold a pointer from new. The held pointer will be deleted automatically if the scope is left without calling `Release` on the [AutoPtr](#) first.

```
#include <dng_auto_ptr.h>
```

#### Public Member Functions

- [AutoPtr](#) ()  
*Construct an [AutoPtr](#) with no referent.*
- [AutoPtr](#) (T \*p)
- [~AutoPtr](#) ()  
*[Reset\(\)](#) is called on destruction.*
- void [Alloc](#) ()  
*Call [Reset](#) with a pointer from new. Uses *T*'s default constructor.*
- T \* [Get](#) () const  
*Return the owned pointer of this [AutoPtr](#), NULL if none. No change in ownership or other effects occur.*

- `T * Release ()`  
*Return the owned pointer of this `AutoPtr`, `NULL` if none. The `AutoPtr` gives up ownership and takes `NULL` as its value.*
- `void Reset (T *p)`  
*If a pointer is owned, it is deleted. Ownership is taken of passed in pointer.*
- `void Reset ()`  
*If a pointer is owned, it is deleted and the `AutoPtr` takes `NULL` as its value.*
- `T * operator → () const`  
*Allows members of the owned pointer to be accessed directly. It is an error to call this if the `AutoPtr` has `NULL` as its value.*
- `T & operator* () const`  
*Returns a reference to the object that the owned pointer points to. It is an error to call this if the `AutoPtr` has `NULL` as its value.*

### 6.1.1 Detailed Description

`template<class T> class AutoPtr< T >`

A class intended to be used in stack scope to hold a pointer from new. The held pointer will be deleted automatically if the scope is left without calling `Release` on the `AutoPtr` first.

### 6.1.2 Constructor & Destructor Documentation

**6.1.2.1** `template<class T> AutoPtr< T >::AutoPtr (T * p) [inline, explicit]`

Construct an `AutoPtr` which owns the argument pointer.

#### Parameters:

- `p` pointer which constructed `AutoPtr` takes ownership of. `p` will be deleted on destruction or `Reset` unless `Release` is called first.

The documentation for this class was generated from the following file:

- [dng\\_auto\\_ptr.h](#)

## 6.2 dng\_1d\_concatenate Class Reference

A [dng\\_1d\\_function](#) that represents the composition (curry) of two other [dng\\_1d\\_](#)-functions.

```
#include <dng_1d_function.h>
```

Inheritance diagram for `dng_1d_concatenate::`

### Public Member Functions

- [dng\\_1d\\_concatenate](#) (const [dng\\_1d\\_function](#) &function1, const [dng\\_1d\\_](#)-[function](#) &function2)
- virtual bool [IsIdentity](#) () const  
*Only true if both function1 and function2 have IsIdentity equal to true.*
- virtual real64 [Evaluate](#) (real64 x) const
- virtual real64 [EvaluateInverse](#) (real64 y) const

### Protected Attributes

- const [dng\\_1d\\_function](#) & **fFunction1**
- const [dng\\_1d\\_function](#) & **fFunction2**

### 6.2.1 Detailed Description

A [dng\\_1d\\_function](#) that represents the composition (curry) of two other [dng\\_1d\\_](#)-functions.

### 6.2.2 Constructor & Destructor Documentation

#### 6.2.2.1 `dng_1d_concatenate::dng_1d_concatenate (const dng\_1d\_function & function1, const dng\_1d\_function & function2)`

Create a [dng\\_1d\\_function](#) which computes  $y = \text{function2.Evaluate}(\text{function1.Evaluate}(x))$ . Compose function1 and function2 to compute  $y = \text{function2.Evaluate}(\text{function1.Evaluate}(x))$ . The range of `function1.Evaluate` must be a subset of 0.0 to 1.0 inclusive, otherwise the result of `function1(x)` will be pinned (clipped) to 0.0 if  $< 0.0$  and to 1.0 if  $> 1.0$ .

**Parameters:**

*function1* Inner function of composition.

*function2* Outer function of composition.

**6.2.3 Member Function Documentation****6.2.3.1 real64 dng\_1d\_concatenate::Evaluate (real64 x) const** [virtual]

Return the composed mapping for value x.

**Parameters:**

*x* A value between 0.0 and 1.0 (inclusive).

**Return values:**

*function2.Evaluate(function1.Evaluate(x)).*

Implements [dng\\_1d\\_function](#).

References [dng\\_1d\\_function::Evaluate\(\)](#).

**6.2.3.2 real64 dng\_1d\_concatenate::EvaluateInverse (real64 y) const**  
[virtual]

Return the reverse mapped value for y. Be careful using this method with compositions where the inner function does not have a range 0.0 to 1.0 . (Or better yet, do not use such functions.)

**Parameters:**

*y* A value to reverse map. Should be within the range of [function2.Evaluate](#).

**Return values:**

A value x such that [function2.Evaluate\(function1.Evaluate\(x\)\)](#) == y (to very close approximation).

Reimplemented from [dng\\_1d\\_function](#).

References [dng\\_1d\\_function::EvaluateInverse\(\)](#).

The documentation for this class was generated from the following files:

- [dng\\_1d\\_function.h](#)
- [dng\\_1d\\_function.cpp](#)

## 6.3 dng\_1d\_function Class Reference

A 1D floating-point function.

```
#include <dng_1d_function.h>
```

Inheritance diagram for dng\_1d\_function::

### Public Member Functions

- virtual bool [IsIdentity](#) () const  
*Returns true if this function is the map  $x \rightarrow y$  such that  $x == y$  for all  $x$ . That is if  $Evaluate(x) == x$  for all  $x$ .*
- virtual real64 [Evaluate](#) (real64 x) const =0
- virtual real64 [EvaluateInverse](#) (real64 y) const

### 6.3.1 Detailed Description

A 1D floating-point function.

The domain (input) is always from 0.0 to 1.0, while the range (output) can be an arbitrary interval.

### 6.3.2 Member Function Documentation

#### 6.3.2.1 virtual real64 dng\_1d\_function::Evaluate (real64 x) const [pure virtual]

Return the mapping for value x. This method must be implemented by a derived class of [dng\\_1d\\_function](#) and the derived class determines the lookup method and function used.

#### Parameters:

*x* A value between 0.0 and 1.0 (inclusive).

#### Return values:

*Mapped* value for x

Implemented in [dng\\_1d\\_identity](#), [dng\\_1d\\_concatenate](#), [dng\\_1d\\_inverse](#), [dng\\_function\\_GammaEncode\\_sRGB](#), [dng\\_function\\_GammaEncode\\_1\\_8](#), [dng\\_function\\_GammaEncode\\_2\\_2](#), [dng\\_noise\\_function](#), [dng\\_function\\_exposure\\_ramp](#), [dng\\_function\\_exposure\\_tone](#), [dng\\_tone\\_curve\\_acr3\\_default](#), and [dng\\_function\\_gamma\\_encode](#).

Referenced by [dng\\_1d\\_concatenate::Evaluate\(\)](#), [dng\\_1d\\_inverse::EvaluateInverse\(\)](#), [EvaluateInverse\(\)](#), [dng\\_color\\_space::GammaEncode\(\)](#), and [dng\\_1d\\_table::Initialize\(\)](#).

### 6.3.2.2 real64 dng\_1d\_function::EvaluateInverse (real64 y) const [virtual]

Return the reverse mapped value for  $y$ . This method can be implemented by derived classes. The default implementation uses Newton's method to solve for  $x$  such that  $\text{Evaluate}(x) == y$ .

#### Parameters:

- $y$  A value to reverse map. Should be within the range of the function implemented by this [dng\\_1d\\_function](#).

#### Return values:

- A value  $x$  such that  $\text{Evaluate}(x) == y$  (to very close approximation).

Reimplemented in [dng\\_1d\\_identity](#), [dng\\_1d\\_concatenate](#), [dng\\_1d\\_inverse](#), [dng\\_function\\_GammaEncode\\_sRGB](#), [dng\\_function\\_GammaEncode\\_1\\_8](#), [dng\\_function\\_GammaEncode\\_2\\_2](#), and [dng\\_tone\\_curve\\_acr3\\_default](#).

References [Evaluate\(\)](#).

Referenced by [dng\\_1d\\_inverse::Evaluate\(\)](#), [dng\\_1d\\_concatenate::EvaluateInverse\(\)](#), and [dng\\_color\\_space::GammaDecode\(\)](#).

The documentation for this class was generated from the following files:

- [dng\\_1d\\_function.h](#)
- [dng\\_1d\\_function.cpp](#)

## 6.4 dng\_1d\_identity Class Reference

An identity ( $x \rightarrow y$  such that  $x == y$  for all  $x$ ) mapping function.

```
#include <dng_1d_function.h>
```

Inheritance diagram for [dng\\_1d\\_identity::](#)

### Public Member Functions

- virtual bool [IsIdentity](#) () const  
*Always returns true for this class.*
- virtual real64 [Evaluate](#) (real64 x) const  
*Always returns x for this class.*
- virtual real64 [EvaluateInverse](#) (real64 y) const  
*Always returns y for this class.*

### Static Public Member Functions

- static const [dng\\_1d\\_function](#) & [Get](#) ()  
*This class is a singleton, and is entirely threadsafe. Use this method to get an instance of the class.*

#### 6.4.1 Detailed Description

An identity ( $x \rightarrow y$  such that  $x == y$  for all  $x$ ) mapping function.

The documentation for this class was generated from the following files:

- [dng\\_1d\\_function.h](#)
- [dng\\_1d\\_function.cpp](#)

## 6.5 dng\_1d\_inverse Class Reference

A [dng\\_1d\\_function](#) that represents the inverse of another [dng\\_1d\\_function](#).

```
#include <dng_1d_function.h>
```

Inheritance diagram for `dng_1d_inverse`:

### Public Member Functions

- [dng\\_1d\\_inverse](#) (const [dng\\_1d\\_function](#) &f)
- virtual bool [IsIdentity](#) () const  
*Returns true if this function is the map  $x \rightarrow y$  such that  $x == y$  for all  $x$ . That is if  $Evaluate(x) == x$  for all  $x$ .*

- virtual `real64 Evaluate` (`real64 x`) `const`
- virtual `real64 EvaluateInverse` (`real64 y`) `const`

### Protected Attributes

- `const dng_1d_function & fFunction`

### 6.5.1 Detailed Description

A `dng_1d_function` that represents the inverse of another `dng_1d_function`.

### 6.5.2 Member Function Documentation

#### 6.5.2.1 `real64 dng_1d_inverse::Evaluate (real64 x) const` [virtual]

Return the mapping for value  $x$ . This method must be implemented by a derived class of `dng_1d_function` and the derived class determines the lookup method and function used.

#### Parameters:

$x$  A value between 0.0 and 1.0 (inclusive).

#### Return values:

*Mapped* value for  $x$

Implements `dng_1d_function`.

References `dng_1d_function::EvaluateInverse()`.

#### 6.5.2.2 `real64 dng_1d_inverse::EvaluateInverse (real64 y) const` [virtual]

Return the reverse mapped value for  $y$ . This method can be implemented by derived classes. The default implementation uses Newton's method to solve for  $x$  such that `Evaluate(x) == y`.

#### Parameters:

$y$  A value to reverse map. Should be within the range of the function implemented by this `dng_1d_function`.

**Return values:**

A value  $x$  such that  $\text{Evaluate}(x) == y$  (to very close approximation).

Reimplemented from [dng\\_1d\\_function](#).

References `dng_1d_function::Evaluate()`.

The documentation for this class was generated from the following files:

- [dng\\_1d\\_function.h](#)
- [dng\\_1d\\_function.cpp](#)

## 6.6 dng\_1d\_table Class Reference

A 1D floating-point lookup table using linear interpolation.

```
#include <dng_1d_table.h>
```

**Public Types**

- enum { **kTableBits** = 12, **kTableSize** = (1 << kTableBits) }  
*Constants denoting size of table.*

**Public Member Functions**

- void [Initialize](#) ([dng\\_memory\\_allocator](#) &allocator, const [dng\\_1d\\_function](#) &function, bool subSample=false)
- real32 [Interpolate](#) (real32  $x$ ) const
- const real32 \* [Table](#) () const  
*Direct access function for table data.*
- void [Expand16](#) (uint16 \*table16) const  
*Expand the table to a 16-bit to 16-bit table.*

**Protected Attributes**

- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fBuffer**
- real32 \* **fTable**

### 6.6.1 Detailed Description

A 1D floating-point lookup table using linear interpolation.

## 6.6.2 Member Function Documentation

### 6.6.2.1 void dng\_1d\_table::Initialize (dng\_memory\_allocator & allocator, const dng\_1d\_function & function, bool subSample = false)

Set up table, initialize entries using function. This method can throw an exception, e.g. if there is not enough memory.

#### Parameters:

*allocator* Memory allocator from which table memory is allocated.

*function* Table is initialized with values of function.Evaluate(0.0) to function.Evaluate(1.0).

*subSample* If true, only sample the function a limited number of times and interpolate.

References dng\_memory\_allocator::Allocate(), dng\_1d\_function::Evaluate(), and AutoPtr< T >::Reset().

### 6.6.2.2 real32 dng\_1d\_table::Interpolate (real32 x) const [inline]

Lookup and interpolate mapping for an input.

#### Parameters:

*x* value from 0.0 to 1.0 used as input for mapping

#### Return values:

*Approximation* of function.Evaluate(x)

References DNG\_ASSERT.

The documentation for this class was generated from the following files:

- [dng\\_1d\\_table.h](#)
- [dng\\_1d\\_table.cpp](#)

## 6.7 dng\_abort\_sniffer Class Reference

Class for signaling user cancellation and receiving progress updates.

```
#include <dng_abort_sniffer.h>
```

### Public Member Functions

- dng\_priority [Priority](#) () const  
*Getter for priority level.*
- void [SetPriority](#) (dng\_priority priority)  
*Setter for priority level.*
- void [SniffNoPriorityWait](#) ()

### Static Public Member Functions

- static void [SniffForAbort](#) (dng\_abort\_sniffer \*sniffer)

### Protected Member Functions

- virtual void [Sniff](#) ()=0
- virtual void [StartTask](#) (const char \*name, real64 fract)
- virtual void [EndTask](#) ()  
*Signals the end of the innermost task that has been started.*
- virtual void [UpdateProgress](#) (real64 fract)

### Friends

- class [dng\\_sniffer\\_task](#)

#### 6.7.1 Detailed Description

Class for signaling user cancellation and receiving progress updates.

DNG SDK clients should derive a host application specific implementation from this class.

#### 6.7.2 Member Function Documentation

##### 6.7.2.1 virtual void dng\_abort\_sniffer::Sniff () [protected, pure virtual]

Should be implemented by derived classes to check for an user cancellation.

Referenced by [SniffForAbort](#)().

### 6.7.2.2 void dng\_abort\_sniffer::SniffForAbort (dng\_abort\_sniffer \* *sniffer*) [static]

Check for pending user cancellation or other abort. ThrowUserCanceled will be called if one is pending. This static method is provided as a convenience for quickly testing for an abort and throwing an exception if one is pending.

#### Parameters:

*sniffer* The dng\_sniffer to test for a pending abort. Can be NULL, in which case there an abort is never signalled.

References Priority(), and Sniff().

Referenced by dng\_stream::Flush(), dng\_stream::Get(), dng\_area\_task::ProcessOnThread(), dng\_stream::Put(), and dng\_sniffer\_task::Sniff().

### 6.7.2.3 void dng\_abort\_sniffer::StartTask (const char \* *name*, real64 *fract*) [protected, virtual]

Signals the start of a named task with processing in the DNG SDK. Tasks may be nested.

#### Parameters:

*name* of the task

*fract* Percentage of total processing this task is expected to take. From 0.0 to 1.0 .

Referenced by dng\_sniffer\_task::dng\_sniffer\_task().

### 6.7.2.4 void dng\_abort\_sniffer::UpdateProgress (real64 *fract*) [protected, virtual]

Signals progress made on current task.

#### Parameters:

*fract* percentage of processing completed on current task. From 0.0 to 1.0 .

Referenced by dng\_sniffer\_task::UpdateProgress().

The documentation for this class was generated from the following files:

- [dng\\_abort\\_sniffer.h](#)
- [dng\\_abort\\_sniffer.cpp](#)

## 6.8 dng\_area\_task Class Reference

Abstract class for rectangular processing operations with support for partitioning across multiple processing resources and observing memory constraints.

```
#include <dng_area_task.h>
```

Inheritance diagram for dng\_area\_task::

### Public Member Functions

- virtual uint32 [MaxThreads](#) () const
- virtual uint32 [MinTaskArea](#) () const
- virtual dng\_point [UnitCell](#) () const
- virtual dng\_point [MaxTileSize](#) () const
- virtual dng\_rect [RepeatingTile1](#) () const
- virtual dng\_rect [RepeatingTile2](#) () const
- virtual dng\_rect [RepeatingTile3](#) () const
- virtual void [Start](#) (uint32 threadCount, const dng\_point &tileSize, [dng\\_memory\\_allocator](#) \*allocator, [dng\\_abort\\_sniffer](#) \*sniffer)
- virtual void [Process](#) (uint32 threadIndex, const dng\_rect &tile, [dng\\_abort\\_sniffer](#) \*sniffer)=0
- virtual void [Finish](#) (uint32 threadCount)
- dng\_point [FindTileSize](#) (const dng\_rect &area) const
- void [ProcessOnThread](#) (uint32 threadIndex, const dng\_rect &area, const dng\_point &tileSize, [dng\\_abort\\_sniffer](#) \*sniffer)

### Static Public Member Functions

- static void [Perform](#) ([dng\\_area\\_task](#) &task, const dng\_rect &area, [dng\\_memory\\_allocator](#) \*allocator, [dng\\_abort\\_sniffer](#) \*sniffer)

### Protected Attributes

- uint32 **fMaxThreads**
- uint32 **fMinTaskArea**
- dng\_point **fUnitCell**
- dng\_point **fMaxTileSize**

### 6.8.1 Detailed Description

Abstract class for rectangular processing operations with support for partitioning across multiple processing resources and observing memory constraints.

### 6.8.2 Member Function Documentation

#### 6.8.2.1 `dng_point dng_area_task::FindTileSize (const dng_rect & area) const`

Find tile size taking into account repeating tiles, unit cell, and maximum tile size.

**Parameters:**

*area* Computation area for which to find tile size.

**Return values:**

*Tile* size as height and width in point.

References `MaxTileSize()`, `RepeatingTile1()`, `RepeatingTile2()`, `RepeatingTile3()`, and `UnitCell()`.

Referenced by `Perform()`.

#### 6.8.2.2 `void dng_area_task::Finish (uint32 threadCount) [virtual]`

Task computation finalization and teardown method. Called after all resources have completed processing. Can be overridden to accumulate results and free resources allocated in `Start`.

**Parameters:**

*threadCount* Number of threads used for processing. Same as value passed to `Start`.

Referenced by `Perform()`.

#### 6.8.2.3 `virtual uint32 dng_area_task::MaxThreads () const [inline, virtual]`

Getter for the maximum number of threads (resources) that can be used for processing

**Return values:**

*Number* of threads, minimum of 1, that can be used for this task.

**6.8.2.4 virtual dng\_point dng\_area\_task::MaxTileSize () const** [inline, virtual]

Getter for maximum size of a tile for processing. Often processing will need to allocate temporary buffers or use other resources that are either fixed or in limited supply. The maximum tile size forces further partitioning if the tile is bigger than this size.

**Return values:**

*Maximum* tile size allowed for this area task.

Referenced by FindTileSize().

**6.8.2.5 virtual uint32 dng\_area\_task::MinTaskArea () const** [inline, virtual]

Getter for minimum area of a partitioned rectangle. Often it is not profitable to use more resources if it requires partitioning the input into chunks that are too small, as the overhead increases more than the speedup. This method can be overridden for a specific task to indicate the smallest area for partitioning. Default is 256x256 pixels.

**Return values:**

*Minimum* area for a partitioned tile in order to give performant operation. (Partitions can be smaller due to small inputs and edge cases.)

**6.8.2.6 void dng\_area\_task::Perform (dng\_area\_task & task, const dng\_rect & area, dng\_memory\_allocator \* allocator, dng\_abort\_sniffer \* sniffer)** [static]

Default resource partitioner that assumes a single resource to be used for processing. Implementations that are aware of multiple processing resources should override (replace) this method. This is usually done in [dng\\_host::PerformAreaTask](#).

**Parameters:**

*task* The task to perform.

*area* The area on which mage processing should be performed.

*allocator* [dng\\_memory\\_allocator](#) to use for allocating temporary buffers, etc.

*sniffer* [dng\\_abort\\_sniffer](#) to use to check for user cancellation and progress updates.

References [FindTileSize\(\)](#), [Finish\(\)](#), [ProcessOnThread\(\)](#), and [Start\(\)](#).

Referenced by [dng\\_host::PerformAreaTask\(\)](#).

#### 6.8.2.7 virtual void dng\_area\_task::Process (uint32 *threadIndex*, const dng\_rect & *tile*, dng\_abort\_sniffer \* *sniffer*) [pure virtual]

Process one tile or fully partitioned area. This method is overridden by derived classes to implement the actual image processing. Note that the sniffer can be ignored if it is certain that a processing task will complete very quickly. This method should never be called directly but rather accessed via [Process](#). There is no allocator parameter as all allocation should be done in [Start](#).

##### Parameters:

*threadIndex* 0 to `threadCount - 1` index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the [Start](#) method.)

*tile* Area to process.

*sniffer* [dng\\_abort\\_sniffer](#) to use to check for user cancellation and progress updates.

Implemented in [dng\\_filter\\_task](#).

Referenced by [ProcessOnThread\(\)](#).

#### 6.8.2.8 void dng\_area\_task::ProcessOnThread (uint32 *threadIndex*, const dng\_rect & *area*, const dng\_point & *tileSize*, dng\_abort\_sniffer \* *sniffer*)

Handle one resource's worth of partitioned tiles. Called after thread partitioning has already been done. Area may be further subdivided to handle maximum tile size, etc. It will be rare to override this method.

##### Parameters:

*threadIndex* 0 to `threadCount - 1` index indicating which thread this is.

*area* Tile area partitioned to this resource.

*tileSize*

*sniffer* [dng\\_abort\\_sniffer](#) to use to check for user cancellation and progress updates.

References [Process\(\)](#), [RepeatingTile1\(\)](#), [RepeatingTile2\(\)](#), [RepeatingTile3\(\)](#), and [dng\\_abort\\_sniffer::SniffForAbort\(\)](#).

Referenced by [Perform\(\)](#).

**6.8.2.9 dng\_rect dng\_area\_task::RepeatingTile1 () const** [virtual]

Getter for [RepeatingTile1](#). [RepeatingTile1](#), [RepeatingTile2](#), and [RepeatingTile3](#) are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final [Process](#) method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A [RepeatingTile](#) value is valid if it is non-empty. Higher numbered [RepeatingTile](#) patterns are only used if all lower ones are non-empty. A [RepeatingTile](#) pattern must be a multiple of [UnitCell](#) in size for all constraints of the [partitionerr](#) to be met.

Referenced by [FindTileSize\(\)](#), and [ProcessOnThread\(\)](#).

**6.8.2.10 dng\_rect dng\_area\_task::RepeatingTile2 () const** [virtual]

Getter for [RepeatingTile2](#). [RepeatingTile1](#), [RepeatingTile2](#), and [RepeatingTile3](#) are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final [Process](#) method is called on will not cross tile boundaries in any of the tile patterns. This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A [RepeatingTile](#) value is valid if it is non-empty. Higher numbered [RepeatingTile](#) patterns are only used if all lower ones are non-empty. A [RepeatingTile](#) pattern must be a multiple of [UnitCell](#) in size for all constraints of the [partitionerr](#) to be met.

Referenced by [FindTileSize\(\)](#), and [ProcessOnThread\(\)](#).

**6.8.2.11 dng\_rect dng\_area\_task::RepeatingTile3 () const** [virtual]

Getter for [RepeatingTile3](#). [RepeatingTile1](#), [RepeatingTile2](#), and [RepeatingTile3](#) are used to establish a set of 0 to 3 tile patterns for which the resulting partitions that the final [Process](#) method is called on will not cross tile boundaries in any of the tile patterns.

This can be used for a processing routine that needs to read from two tiles and write to a third such that all the tiles are aligned and sized in a certain way. A RepeatingTile value is valid if it is non-empty. Higher numbered RepeatingTile patterns are only used if all lower ones are non-empty. A RepeatingTile pattern must be a multiple of UnitCell in size for all constraints of the partitionerr to be met.

Referenced by FindTileSize(), and ProcessOnThread().

**6.8.2.12** `void dng_area_task::Start (uint32 threadCount, const dng_point & tileSize, dng_memory_allocator * allocator, dng_abort_sniffer * sniffer)` [virtual]

Task startup method called before any processing is done on partitions. The Start method is called before any processing is done and can be overridden to allocate temporary buffers, etc.

**Parameters:**

*threadCount* Total number of threads that will be used for processing. Less than or equal to MaxThreads.

*tileSize* Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)

*allocator* [dng\\_memory\\_allocator](#) to use for allocating temporary buffers, etc.

*sniffer* Sniffer to test for user cancellation and to set up progress.

Reimplemented in [dng\\_filter\\_task](#).

Referenced by Perform().

**6.8.2.13** `virtual dng_point dng_area_task::UnitCell () const` [inline, virtual]

Getter for dimensions of which partitioned tiles should be a multiple. Various methods of processing prefer certain alignments. The partitioning attempts to construct tiles such that the sizes are a multiple of the dimensions of this point.

**Return values:**

*a* point giving preferred alignment in x and y

Referenced by FindTileSize().

The documentation for this class was generated from the following files:

- [dng\\_area\\_task.h](#)
- [dng\\_area\\_task.cpp](#)

## 6.9 dng\_camera\_profile Class Reference

Container for DNG camera color profile and calibration data.

```
#include <dng_camera_profile.h>
```

### Public Member Functions

- void [SetName](#) (const char \*name)
- const [dng\\_string](#) & [Name](#) () const
- bool [NameIsEmbedded](#) () const
- void [SetCalibrationIlluminant1](#) (uint32 light)
- void [SetCalibrationIlluminant2](#) (uint32 light)
- uint32 [CalibrationIlluminant1](#) () const
- uint32 [CalibrationIlluminant2](#) () const
- real64 [CalibrationTemperature1](#) () const
- real64 [CalibrationTemperature2](#) () const
- void [SetColorMatrix1](#) (const [dng\\_matrix](#) &m)
- void [SetColorMatrix2](#) (const [dng\\_matrix](#) &m)
- bool [HasColorMatrix1](#) () const
  - Predicate to test if first camera matrix is set.*
- bool [HasColorMatrix2](#) () const
  - Predicate to test if second camera matrix is set.*
- const [dng\\_matrix](#) & [ColorMatrix1](#) () const
  - Getter for first of up to two color matrices used for calibrations.*
- const [dng\\_matrix](#) & [ColorMatrix2](#) () const
  - Getter for second of up to two color matrices used for calibrations.*
- void [SetForwardMatrix1](#) (const [dng\\_matrix](#) &m)
  - Setter for first of up to two forward matrices used for calibrations.*
- void [SetForwardMatrix2](#) (const [dng\\_matrix](#) &m)
  - Setter for second of up to two forward matrices used for calibrations.*
- const [dng\\_matrix](#) & [ForwardMatrix1](#) () const
  - Getter for first of up to two forward matrices used for calibrations.*

- const dng\_matrix & **ForwardMatrix2** () const  
*Getter for second of up to two forward matrices used for calibrations.*
- void **SetReductionMatrix1** (const dng\_matrix &m)
- void **SetReductionMatrix2** (const dng\_matrix &m)
- const dng\_matrix & **ReductionMatrix1** () const  
*Getter for first of up to two dimensionality reduction hints for four color cameras.*
- const dng\_matrix & **ReductionMatrix2** () const  
*Getter for second of up to two dimensionality reduction hints for four color cameras.*
- const **dng\_fingerprint** & **Fingerprint** () const  
*Getter function from profile fingerprint.*
- dng\_camera\_profile\_id **ProfileID** () const
- void **SetCopyright** (const char \*copyright)
- const dng\_string & **Copyright** () const
- void **SetEmbedPolicy** (uint32 policy)
- uint32 **EmbedPolicy** () const
- bool **IsLegalToEmbed** () const
- bool **HasHueSatDeltas** () const
- const dng\_hue\_sat\_map & **HueSatDeltas1** () const
- void **SetHueSatDeltas1** (const dng\_hue\_sat\_map &deltas1)
- const dng\_hue\_sat\_map & **HueSatDeltas2** () const
- void **SetHueSatDeltas2** (const dng\_hue\_sat\_map &deltas2)
- bool **HasLookTable** () const
- const dng\_hue\_sat\_map & **LookTable** () const
- void **SetLookTable** (const dng\_hue\_sat\_map &table)
- const dng\_tone\_curve & **ToneCurve** () const
- void **SetToneCurve** (const dng\_tone\_curve &curve)
- void **SetProfileCalibrationSignature** (const char \*signature)
- const dng\_string & **ProfileCalibrationSignature** () const
- void **SetUniqueCameraModelRestriction** (const char \*camera)
- const dng\_string & **UniqueCameraModelRestriction** () const
- void **SetWasReadFromDNG** (bool state=true)
- bool **WasReadFromDNG** () const
- bool **IsValid** (uint32 channels) const
- bool **EqualData** (const **dng\_camera\_profile** &profile) const
- void **Parse** (**dng\_stream** &stream, **dng\_camera\_profile\_info** &profileInfo)  
*Parse profile from dng\_camera\_profile\_info data.*
- bool **ParseExtended** (**dng\_stream** &stream)

- virtual void [SetFourColorBayer](#) ()  
*Convert from a three-color to a four-color Bayer profile.*
- dng\_hue\_sat\_map \* [HueSatMapForWhite](#) (const dng\_xy\_coord &white) const
- void [Stub](#) ()  
*Stub out the profile (free memory used by large tables).*
- bool [WasStubbed](#) () const  
*Was this profile stubbed?*

### Static Public Member Functions

- static void [NormalizeColorMatrix](#) (dng\_matrix &m)  
*Utility function to normalize the scale of the color matrix.*
- static void [NormalizeForwardMatrix](#) (dng\_matrix &m)  
*Utility function to normalize the scale of the forward matrix.*

### Protected Member Functions

- void [ClearFingerprint](#) ()
- void [CalculateFingerprint](#) () const

### Static Protected Member Functions

- static real64 [IlluminantToTemperature](#) (uint32 light)
- static bool [ValidForwardMatrix](#) (const dng\_matrix &m)
- static void [ReadHueSatMap](#) (dng\_stream &stream, dng\_hue\_sat\_map &hueSatMap, uint32 hues, uint32 sats, uint32 vals, bool skipSat0)

### Protected Attributes

- dng\_string [fName](#)
- uint32 [fCalibrationIlluminant1](#)
- uint32 [fCalibrationIlluminant2](#)
- dng\_matrix [fColorMatrix1](#)
- dng\_matrix [fColorMatrix2](#)
- dng\_matrix [fForwardMatrix1](#)
- dng\_matrix [fForwardMatrix2](#)
- dng\_matrix [fReductionMatrix1](#)

- dng\_matrix **fReductionMatrix2**
- [dng\\_fingerprint](#) **fFingerprint**
- dng\_string **fCopyright**
- uint32 **fEmbedPolicy**
- dng\_hue\_sat\_map **fHueSatDeltas1**
- dng\_hue\_sat\_map **fHueSatDeltas2**
- dng\_hue\_sat\_map **fLookTable**
- dng\_tone\_curve **fToneCurve**
- dng\_string **fProfileCalibrationSignature**
- dng\_string **fUniqueCameraModelRestriction**
- bool **fWasReadFromDNG**
- bool **fWasStubbed**

### 6.9.1 Detailed Description

Container for DNG camera color profile and calibration data.

### 6.9.2 Member Function Documentation

#### 6.9.2.1 uint32 dng\_camera\_profile::CalibrationIlluminant1 () const [inline]

Getter for first of up to two light sources used for calibration. Uses the EXIF encodings for illuminant and is used to distinguish which matrix to use. Corresponds to the DNG CalibrationIlluminant1 tag.

Referenced by CalibrationTemperature1().

#### 6.9.2.2 uint32 dng\_camera\_profile::CalibrationIlluminant2 () const [inline]

Getter for second of up to two light sources used for calibration. Uses the EXIF encodings for illuminant and is used to distinguish which matrix to use. Corresponds to the DNG CalibrationIlluminant2 tag.

Referenced by CalibrationTemperature2().

#### 6.9.2.3 real64 dng\_camera\_profile::CalibrationTemperature1 () const [inline]

Getter for first of up to two light sources used for calibration, returning result as color temperature.

References `CalibrationIlluminant1()`.

Referenced by `dng_color_spec::dng_color_spec()`, and `HueSatMapForWhite()`.

#### 6.9.2.4 `real64 dng_camera_profile::CalibrationTemperature2 () const` [inline]

Getter for second of up to two light sources used for calibration, returning result as color temperature.

References `CalibrationIlluminant2()`.

Referenced by `dng_color_spec::dng_color_spec()`, and `HueSatMapForWhite()`.

#### 6.9.2.5 `const dng_string& dng_camera_profile::Copyright () const` [inline]

Getter for camera profile copyright.

##### Return values:

*Copyright* string for profile.

#### 6.9.2.6 `bool dng_camera_profile::EqualData (const dng_camera_profile & profile) const`

Predicate to check if two camera profiles are colorwise equal, thus ignores the profile name.

##### Parameters:

*profile* Camera profile to compare to.

#### 6.9.2.7 `dng_hue_sat_map * dng_camera_profile::HueSatMapForWhite (const dng_xy_coord & white) const`

Find the hue/sat table to use for a given white point, if any. The calling routine owns the resulting table.

References CalibrationTemperature1(), and CalibrationTemperature2().

#### 6.9.2.8 bool dng\_camera\_profile::IsValid (uint32 *channels*) const

Determines if this a valid profile for this number of color channels?

**Return values:**

*true* if the profile is valid.

Referenced by dng\_color\_spec::dng\_color\_spec(), dng\_info::Parse(), and SetFourColorBayer().

#### 6.9.2.9 const dng\_string& dng\_camera\_profile::Name () const [inline]

Getter for camera profile name.

**Return values:**

*Name* of profile.

Referenced by ProfileID().

#### 6.9.2.10 bool dng\_camera\_profile::NameIsEmbedded () const [inline]

Test if this name is embedded.

**Return values:**

*true* if the name matches the name of the embedded camera profile.

#### 6.9.2.11 bool dng\_camera\_profile::ParseExtended (dng\_stream & *stream*)

Parse from an extended profile stream, which is similar to stand alone TIFF file.

References Parse().

**6.9.2.12 dng\_camera\_profile\_id dng\_camera\_profile::ProfileID () const**  
[inline]

Getter for camera profile id.

**Return values:**

*ID* of profile.

References Fingerprint(), and Name().

**6.9.2.13 void dng\_camera\_profile::SetCalibrationIlluminant1 (uint32 light)**  
[inline]

Setter for first of up to two light sources used for calibration. Uses the EXIF encodings for illuminant and is used to distinguish which matrix to use. Corresponds to the DNG CalibrationIlluminant1 tag.

Referenced by Parse().

**6.9.2.14 void dng\_camera\_profile::SetCalibrationIlluminant2 (uint32 light)**  
[inline]

Setter for second of up to two light sources used for calibration. Uses the EXIF encodings for illuminant and is used to distinguish which matrix to use. Corresponds to the DNG CalibrationIlluminant2 tag.

Referenced by Parse().

**6.9.2.15 void dng\_camera\_profile::SetColorMatrix1 (const dng\_matrix & m)**

Setter for first of up to two color matrices used for reference camera calibrations. These matrices map XYZ values to camera values. The DNG SDK needs to map colors that direction in order to determine the clipping points for highlight recovery logic based on the white point. If cameras were all three-color, the matrix could be stored as a forward matrix. The inverse matrix is required to support four-color cameras.

References NormalizeColorMatrix().

Referenced by Parse().

**6.9.2.16 void dng\_camera\_profile::SetColorMatrix2 (const dng\_matrix & m)**

Setter for second of up to two color matrices used for reference camera calibrations. These matrices map XYZ values to camera values. The DNG SDK needs to map colors that direction in order to determine the clipping points for highlight recovery logic based on the white point. If cameras were all three-color, the matrix could be stored as a forward matrix. The inverse matrix is required to support four-color cameras.

References NormalizeColorMatrix().

Referenced by Parse().

**6.9.2.17 void dng\_camera\_profile::SetCopyright (const char \* *copyright*)**  
[inline]

Setter for camera profile copyright.

**Parameters:**

*copyright* Copyright string to use for this camera profile.

Referenced by Parse().

**6.9.2.18 void dng\_camera\_profile::SetName (const char \* *name*)** [inline]

Setter for camera profile name.

**Parameters:**

*name* Name to use for this camera profile.

Referenced by Parse().

**6.9.2.19 void dng\_camera\_profile::SetReductionMatrix1 (const dng\_matrix & m)**

Setter for first of up to two dimensionality reduction hints for four-color cameras. This is an optional matrix that maps four components to three. See Appendix 6 of the [DNG 1.1.0 specification](#).

Referenced by Parse().

### 6.9.2.20 void dng\_camera\_profile::SetReductionMatrix2 (const dng\_matrix & m)

Setter for second of up to two dimensionality reduction hints for four-color cameras. This is an optional matrix that maps four components to three. See Appendix 6 of the [DNG 1.1.0 specification](#).

Referenced by Parse().

### 6.9.2.21 void dng\_camera\_profile::SetUniqueCameraModelRestriction (const char \* camera) [inline]

Setter for camera unique model name to restrict use of this profile.

#### Parameters:

*camera* Camera unique model name designating only camera this profile can be used with. (Empty string for no restriction.)

Referenced by Parse().

### 6.9.2.22 const dng\_string& dng\_camera\_profile::UniqueCameraModelRestriction () const [inline]

Getter for camera unique model name to restrict use of this profile.

#### Return values:

*Unique* model name of only camera this profile can be used with or empty if no restriction.

The documentation for this class was generated from the following files:

- [dng\\_camera\\_profile.h](#)
- [dng\\_camera\\_profile.cpp](#)

## 6.10 dng\_color\_space Class Reference

An abstract color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng\_color\_space::

### Public Member Functions

- const dng\_matrix & **MatrixToPCS** () const  
*Return a matrix which transforms source data in this color space into the Profile Connection Space.*
- const dng\_matrix & **MatrixFromPCS** () const  
*Return a matrix which transforms Profile Connection Space data into this color space.*
- bool **IsMonochrome** () const  
*Predicate which is true if this color space is monochrome (has only a single column).*
- virtual const dng\_1d\_function & **GammaFunction** () const  
*Getter for the gamma function for this color space.*
- bool **IsLinear** () const  
*Returns true if this color space is linear. (I.e. has gamma 1.0).*
- real64 **GammaEncode** (real64 x) const  
*Map an input value through this color space's encoding gamma.*
- real64 **GammaDecode** (real64 y) const  
*Map an input value through this color space's decoding gamma (inverse of the encoding gamma).*
- virtual bool **ICCProfile** (uint32 &size, const uint8 \*&data) const

### Protected Member Functions

- void **SetMonochrome** ()
- void **SetMatrixToPCS** (const dng\_matrix\_3by3 &M)

### Protected Attributes

- dng\_matrix **fMatrixToPCS**
- dng\_matrix **fMatrixFromPCS**

### 6.10.1 Detailed Description

An abstract color space.

### 6.10.2 Member Function Documentation

#### 6.10.2.1 `bool dng_color_space::ICCProfile (uint32 & size, const uint8 *& data) const` [virtual]

Getter for ICC profile, if this color space has one.

##### Parameters:

- size* Out parameter which receives size on return.
- data* Receives bytes of profile.

##### Return values:

*Returns* true if this color space has an ICC profile, false otherwise.

Reimplemented in [dng\\_space\\_sRGB](#), [dng\\_space\\_AdobeRGB](#), [dng\\_space\\_ColorMatch](#), [dng\\_space\\_ProPhoto](#), [dng\\_space\\_GrayGamma18](#), and [dng\\_space\\_GrayGamma22](#).

Referenced by `dng_image_writer::WriteTIFF()`.

The documentation for this class was generated from the following files:

- [dng\\_color\\_space.h](#)
- [dng\\_color\\_space.cpp](#)

## 6.11 dng\_color\_spec Class Reference

```
#include <dng_color_spec.h>
```

### Public Member Functions

- `dng_color_spec` (const [dng\\_negative](#) &negative, const [dng\\_camera\\_profile](#) \*profile)
- `uint32 Channels` () const
- `void SetWhiteXY` (const [dng\\_xy\\_coord](#) &white)
- `const dng_xy_coord & WhiteXY` () const
- `const dng_vector & CameraWhite` () const
- `const dng_matrix & CameraToPCS` () const
- `dng_xy_coord NeutralToXY` (const [dng\\_vector](#) &neutral)

### 6.11.1 Detailed Description

Color transform taking into account white point and camera calibration and individual calibration from DNG negative.

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 `dng_color_spec::dng_color_spec (const dng_negative & negative, const dng_camera_profile * profile)`

Read calibration info from DNG negative and construct a `dng_color_spec`.

References `dng_negative::AnalogBalance()`, `dng_camera_profile::CalibrationTemperature1()`, `dng_camera_profile::CalibrationTemperature2()`, `dng_negative::CameraCalibration1()`, `dng_negative::CameraCalibration2()`, `dng_camera_profile::ColorMatrix1()`, `dng_camera_profile::ColorMatrix2()`, `dng_camera_profile::ForwardMatrix1()`, `dng_camera_profile::ForwardMatrix2()`, `dng_camera_profile::HasColorMatrix2()`, `dng_camera_profile::IsValid()`, `dng_camera_profile::NormalizeForwardMatrix()`, `dng_camera_profile::ReductionMatrix1()`, `dng_camera_profile::ReductionMatrix2()`, `ThrowBadFormat()`, `ThrowProgramError()`, and `dng_camera_profile::WasStubbed()`.

### 6.11.3 Member Function Documentation

#### 6.11.3.1 `const dng_matrix & dng_color_spec::CameraToPCS () const`

Getter for camera to Profile Connection Space color transform.

#### Return values:

A transform that takes into account all camera calibration transforms and white point.

References `DNG_ASSERT`.

#### 6.11.3.2 `const dng_vector & dng_color_spec::CameraWhite () const`

Return white point in camera native color coordinates.

**Return values:**

A `dng_vector` with components ranging from 0.0 to 1.0 that is normalized such that one component is equal to 1.0 .

References DNG\_ASSERT.

**6.11.3.3** `uint32 dng_color_spec::Channels () const` [inline]

Number of channels used for this color transform. Three for most cameras.

**6.11.3.4** `dng_xy_coord dng_color_spec::NeutralToXY (const dng_vector & neutral)`

Return the XY value to use for SetWhiteXY for a given camera color space coordinate as the white point.

**Parameters:**

*neutral* A camera color space value to use for white point. Components range from 0.0 to 1.0 and should be normalized such that the largest value is 1.0 .

**Return values:**

*White* point in XY space that makes neutral map to this XY value as closely as possible.

**6.11.3.5** `void dng_color_spec::SetWhiteXY (const dng_xy_coord & white)`

Setter for white point. Value is as XY colorspace coordinate.

**Parameters:**

*white* White point to set as an XY value.

**6.11.3.6** `const dng_xy_coord & dng_color_spec::WhiteXY () const`

Getter for white point. Value is as XY colorspace coordinate.

**Return values:**

*XY* value of white point.

References DNG\_ASSERT.

The documentation for this class was generated from the following files:

- [dng\\_color\\_spec.h](#)
- [dng\\_color\\_spec.cpp](#)

## 6.12 dng\_const\_tile\_buffer Class Reference

Class to get resource acquisition is instantiation behavior for constant (read-only) tile buffers.

```
#include <dng_image.h>
```

Inheritance diagram for dng\_const\_tile\_buffer::

**Public Member Functions**

- [dng\\_const\\_tile\\_buffer](#) (const [dng\\_image](#) &image, const [dng\\_rect](#) &tile)

### 6.12.1 Detailed Description

Class to get resource acquisition is instantiation behavior for constant (read-only) tile buffers.

### 6.12.2 Constructor & Destructor Documentation

#### 6.12.2.1 dng\_const\_tile\_buffer::dng\_const\_tile\_buffer (const [dng\\_image](#) & *image*, const [dng\\_rect](#) & *tile*)

Obtain a read-only tile from an image.

**Parameters:**

*image* Image tile will come from.

*tile* Rectangle denoting extent of tile.

The documentation for this class was generated from the following files:

- [dng\\_image.h](#)
- [dng\\_image.cpp](#)

## 6.13 dng\_date\_time Class Reference

Class for holding a date/time and converting to and from relevant date/time formats.

```
#include <dng_date_time.h>
```

### Public Member Functions

- [dng\\_date\\_time](#) ()  
*Construct an invalid date/time.*
- [dng\\_date\\_time](#) (uint32 year, uint32 month, uint32 day, uint32 hour, uint32 minute, uint32 second)
- bool [IsValid](#) () const
- bool [NotValid](#) () const
- bool [operator==](#) (const [dng\\_date\\_time](#) &dt) const  
*Equal operator.*
- bool [operator!=](#) (const [dng\\_date\\_time](#) &dt) const
- void [Clear](#) ()  
*Set date to an invalid value.*
- bool [Parse](#) (const char \*s)

### Public Attributes

- uint32 **fYear**
- uint32 **fMonth**
- uint32 **fDay**
- uint32 **fHour**
- uint32 **fMinute**
- uint32 **fSecond**

#### 6.13.1 Detailed Description

Class for holding a date/time and converting to and from relevant date/time formats.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 `dng_date_time::dng_date_time (uint32 year, uint32 month, uint32 day, uint32 hour, uint32 minute, uint32 second)`

Construct a date/time with specific values.

**Parameters:**

*year* Year to use as actual integer value, such as 2006.

*month* Month to use from 1 - 12, where 1 is January.

*day* Day of month to use from 1 -31, where 1 is the first.

*hour* Hour of day to use from 0 - 23, where 0 is midnight.

*minute* Minute of hour to use from 0 - 59.

*second* Second of minute to use from 0 - 59.

### 6.13.3 Member Function Documentation

#### 6.13.3.1 `bool dng_date_time::IsValid () const`

Predicate to determine if a date is valid.

**Return values:**

*true* if all fields are within range.

Referenced by `LocalTimeZone()`, `NotValid()`, and `Parse()`.

#### 6.13.3.2 `bool dng_date_time::NotValid () const` `[inline]`

Predicate to determine if a date is invalid.

**Return values:**

*true* if any field is out of range.

References `IsValid()`.

### 6.13.3.3 bool dng\_date\_time::Parse (const char \* s)

Parse an EXIF format date string.

#### Parameters:

- s Input date string to parse.

#### Return values:

- true* if date was parsed successfully and date is valid.

References IsValid().

The documentation for this class was generated from the following files:

- [dng\\_date\\_time.h](#)
- dng\_date\_time.cpp

## 6.14 dng\_date\_time\_info Class Reference

Class for holding complete data/time/zone information.

```
#include <dng_date_time.h>
```

#### Public Member Functions

- bool **IsValid** () const
- bool **NotValid** () const
- void **Clear** ()
- const [dng\\_date\\_time](#) & **DateTime** () const
- void **SetDateTime** (const [dng\\_date\\_time](#) &dt)
- const dng\_string & **Subseconds** () const
- void **SetSubseconds** (const dng\_string &s)
- const [dng\\_time\\_zone](#) & **TimeZone** () const
- void **SetZone** (const [dng\\_time\\_zone](#) &zone)
- void **Decode\_ISO\_8601** (const char \*s)
- dng\_string **Encode\_ISO\_8601** () const
- void **Decode\_IPTC\_Date** (const char \*s)
- dng\_string **Encode\_IPTC\_Date** () const
- void **Decode\_IPTC\_Time** (const char \*s)
- dng\_string **Encode\_IPTC\_Time** () const

### 6.14.1 Detailed Description

Class for holding complete data/time/zone information.

The documentation for this class was generated from the following files:

- [dng\\_date\\_time.h](#)
- [dng\\_date\\_time.cpp](#)

## 6.15 `dng_date_time_storage_info` Class Reference

Store file offset from which date was read.

```
#include <dng_date_time.h>
```

### Public Member Functions

- [dng\\_date\\_time\\_storage\\_info](#) ()  
*The default constructor initializes to an invalid state.*
- [dng\\_date\\_time\\_storage\\_info](#) (uint64 offset, [dng\\_date\\_time\\_format](#) format)  
*Construct with file offset and date format.*
- [bool IsValid](#) () const
- [uint64 Offset](#) () const
- [dng\\_date\\_time\\_format Format](#) () const

### 6.15.1 Detailed Description

Store file offset from which date was read.

Used internally by Adobe to update date in original file.

#### Warning:

Use at your own risk.

### 6.15.2 Member Function Documentation

#### 6.15.2.1 `dng_date_time_format dng_date_time_storage_info::Format` () const

Get for format date was originally stored in file. Throws a `dng_error_unknown` exception if offset is invalid.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_unknown` if offset is not valid.

References `IsValid()`, and `ThrowProgramError()`.

**6.15.2.2 `bool dng_date_time_storage_info::IsValid () const`**

Predicate to determine if an offset is valid.

**Return values:**

*true* if offset is valid.

Referenced by `Format()`, and `Offset()`.

**6.15.2.3 `uint64 dng_date_time_storage_info::Offset () const`**

Getter for offset in file.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_unknown` if offset is not valid.

References `IsValid()`, and `ThrowProgramError()`.

The documentation for this class was generated from the following files:

- [dng\\_date\\_time.h](#)
- [dng\\_date\\_time.cpp](#)

**6.16 `dng_dirty_tile_buffer` Class Reference**

Class to get resource acquisition is instantiation behavior for dirty (writable) tile buffers.

```
#include <dng_image.h>
```

Inheritance diagram for `dng_dirty_tile_buffer::`

## Public Member Functions

- [dng\\_dirty\\_tile\\_buffer](#) ([dng\\_image](#) &image, const [dng\\_rect](#) &tile)

### 6.16.1 Detailed Description

Class to get resource acquisition is instantiation behavior for dirty (writable) tile buffers.

### 6.16.2 Constructor & Destructor Documentation

#### 6.16.2.1 `dng_dirty_tile_buffer::dng_dirty_tile_buffer` ([dng\\_image](#) & *image*, const [dng\\_rect](#) & *tile*)

Obtain a writable tile from an image.

#### Parameters:

- image* Image tile will come from.
- tile* Rectangle denoting extent of tile.

The documentation for this class was generated from the following files:

- [dng\\_image.h](#)
- [dng\\_image.cpp](#)

## 6.17 `dng_exception` Class Reference

All exceptions thrown by the DNG SDK use this exception class.

```
#include <dng_exceptions.h>
```

## Public Member Functions

- [dng\\_exception](#) ([dng\\_error\\_code](#) code)
- [dng\\_error\\_code](#) `ErrorCode` () const

### 6.17.1 Detailed Description

All exceptions thrown by the DNG SDK use this exception class.

## 6.17.2 Constructor & Destructor Documentation

### 6.17.2.1 dng\_exception::dng\_exception (dng\_error\_code code) [inline]

Construct an exception representing the given error code.

#### Parameters:

*code* Error code this exception is for.

## 6.17.3 Member Function Documentation

### 6.17.3.1 dng\_error\_code dng\_exception::ErrorCode () const [inline]

Getter for error code of this exception

#### Return values:

*The* error code of this exception.

The documentation for this class was generated from the following file:

- [dng\\_exceptions.h](#)

## 6.18 dng\_exif Class Reference

Container class for parsing and holding EXIF tags.

```
#include <dng_exif.h>
```

### Public Member Functions

- virtual [dng\\_exif](#) \* **Clone** () const
- void **SetExposureTime** (real64 et, bool snap=true)
- void **SetShutterSpeedValue** (real64 ss)
- void **SetFNumber** (real64 fs)
- void **SetApertureValue** (real64 av)
- void **UpdateDateTime** (const [dng\\_date\\_time\\_info](#) &dt)
- virtual bool **ParseTag** ([dng\\_stream](#) &stream, [dng\\_shared](#) &shared, uint32 parentCode, bool isMainIFD, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual void **PostParse** ([dng\\_host](#) &host, [dng\\_shared](#) &shared)

### Static Public Member Functions

- static real64 **SnapExposureTime** (real64 et)
- static dng\_urational **EncodeFNumber** (real64 fs)

### Public Attributes

- dng\_string **fImageDescription**
- dng\_string **fMake**
- dng\_string **fModel**
- dng\_string **fSoftware**
- dng\_string **fArtist**
- dng\_string **fCopyright**
- dng\_string **fCopyright2**
- dng\_string **fUserComment**
- [dng\\_date\\_time\\_info](#) **fDateTime**
- [dng\\_date\\_time\\_storage\\_info](#) **fDateTimeStorageInfo**
- [dng\\_date\\_time\\_info](#) **fDateTimeOriginal**
- [dng\\_date\\_time\\_storage\\_info](#) **fDateTimeOriginalStorageInfo**
- [dng\\_date\\_time\\_info](#) **fDateTimeDigitized**
- [dng\\_date\\_time\\_storage\\_info](#) **fDateTimeDigitizedStorageInfo**
- uint32 **fTIFF\_EP\_StandardID**
- uint32 **fExifVersion**
- uint32 **fFlashPixVersion**
- dng\_urational **fExposureTime**
- dng\_urational **fFNumber**
- dng\_srational **fShutterSpeedValue**
- dng\_urational **fApertureValue**
- dng\_srational **fBrightnessValue**
- dng\_srational **fExposureBiasValue**
- dng\_urational **fMaxApertureValue**
- dng\_urational **fFocalLength**
- dng\_urational **fDigitalZoomRatio**
- dng\_urational **fExposureIndex**
- dng\_urational **fSubjectDistance**
- dng\_urational **fGamma**
- dng\_urational **fBatteryLevelR**
- dng\_string **fBatteryLevelA**
- uint32 **fExposureProgram**
- uint32 **fMeteringMode**
- uint32 **fLightSource**
- uint32 **fFlash**
- uint32 **fFlashMask**

- uint32 **fSensingMethod**
- uint32 **fColorSpace**
- uint32 **fFileSource**
- uint32 **fSceneType**
- uint32 **fCustomRendered**
- uint32 **fExposureMode**
- uint32 **fWhiteBalance**
- uint32 **fSceneCaptureType**
- uint32 **fGainControl**
- uint32 **fContrast**
- uint32 **fSaturation**
- uint32 **fSharpness**
- uint32 **fSubjectDistanceRange**
- uint32 **fSelfTimerMode**
- uint32 **fImageNumber**
- uint32 **fFocalLengthIn35mmFilm**
- uint32 **fISOSpeedRatings** [3]
- uint32 **fSubjectAreaCount**
- uint32 **fSubjectArea** [4]
- uint32 **fComponentsConfiguration**
- dng\_urational **fCompressedBitsPerPixel**
- uint32 **fPixelXDimension**
- uint32 **fPixelYDimension**
- dng\_urational **fFocalPlaneXResolution**
- dng\_urational **fFocalPlaneYResolution**
- uint32 **fFocalPlaneResolutionUnit**
- uint32 **fCFARepeatPatternRows**
- uint32 **fCFARepeatPatternCols**
- uint8 **fCFAPattern** [[kMaxCFAPattern](#)][[kMaxCFAPattern](#)]
- [dng\\_fingerprint](#) **fImageUniqueID**
- uint32 **fGPSVersionID**
- dng\_string **fGPSLatitudeRef**
- dng\_urational **fGPSLatitude** [3]
- dng\_string **fGPSLongitudeRef**
- dng\_urational **fGPSLongitude** [3]
- uint32 **fGPSAltitudeRef**
- dng\_urational **fGPSAltitude**
- dng\_urational **fGPSTimeStamp** [3]
- dng\_string **fGPSSatellites**
- dng\_string **fGPSStatus**
- dng\_string **fGPSMeasureMode**
- dng\_urational **fGPSDOP**
- dng\_string **fGPSSpeedRef**

- dng\_urational **fGPSSpeed**
- dng\_string **fGPSTrackRef**
- dng\_urational **fGPSTrack**
- dng\_string **fGPSImgDirectionRef**
- dng\_urational **fGPSImgDirection**
- dng\_string **fGPSMapDatum**
- dng\_string **fGPSDestLatitudeRef**
- dng\_urational **fGPSDestLatitude** [3]
- dng\_string **fGPSDestLongitudeRef**
- dng\_urational **fGPSDestLongitude** [3]
- dng\_string **fGPSDestBearingRef**
- dng\_urational **fGPSDestBearing**
- dng\_string **fGPSDestDistanceRef**
- dng\_urational **fGPSDestDistance**
- dng\_string **fGPSProcessingMethod**
- dng\_string **fGPSAreaInformation**
- dng\_string **fGPSDateStamp**
- uint32 **fGPSDifferential**
- dng\_string **fInteroperabilityIndex**
- uint32 **fInteroperabilityVersion**
- dng\_string **fRelatedImageFileFormat**
- uint32 **fRelatedImageWidth**
- uint32 **fRelatedImageLength**
- dng\_string **fCameraSerialNumber**
- dng\_urational **fLensInfo** [4]
- dng\_string **fLensID**
- dng\_string **fLensName**
- dng\_string **fLensSerialNumber**
- dng\_urational **fFlashCompensation**
- dng\_string **fOwnerName**
- dng\_string **fFirmware**

### Protected Member Functions

- virtual bool **Parse\_ifd0** (dng\_stream &stream, dng\_shared &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual bool **Parse\_ifd0\_main** (dng\_stream &stream, dng\_shared &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual bool **Parse\_ifd0\_exif** (dng\_stream &stream, dng\_shared &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)

- virtual bool **Parse\_gps** (`dng_stream` &stream, `dng_shared` &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual bool **Parse\_interoperability** (`dng_stream` &stream, `dng_shared` &shared, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)

### 6.18.1 Detailed Description

Container class for parsing and holding EXIF tags.

Public member fields are documented in [EXIF specification](#).

The documentation for this class was generated from the following files:

- [dng\\_exif.h](#)
- [dng\\_exif.cpp](#)

## 6.19 `dng_file_stream` Class Reference

A stream to/from a disk file. See [dng\\_stream](#) for read/write interface.

```
#include <dng_file_stream.h>
```

Inheritance diagram for `dng_file_stream`::

### Public Member Functions

- [dng\\_file\\_stream](#) (const char \*filename, bool output=false, uint32 bufferSize=kDefaultBufferSize)

### Protected Member Functions

- virtual uint64 **DoGetLength** ()
- virtual void **DoRead** (void \*data, uint32 count, uint64 offset)
- virtual void **DoWrite** (const void \*data, uint32 count, uint64 offset)

### 6.19.1 Detailed Description

A stream to/from a disk file. See [dng\\_stream](#) for read/write interface.

## 6.19.2 Constructor & Destructor Documentation

### 6.19.2.1 `dng_file_stream::dng_file_stream (const char *filename, bool output = false, uint32 bufferSize = kDefaultBufferSize)`

Open a stream on a file.

#### Parameters:

- filename* Pathname in platform syntax.
- output* Set to true if writing, false otherwise.
- bufferSize* size of internal buffer to use. Defaults to 4k.

References `ThrowOpenFile()`, and `ThrowSilentError()`.

The documentation for this class was generated from the following files:

- [dng\\_file\\_stream.h](#)
- [dng\\_file\\_stream.cpp](#)

## 6.20 `dng_filter_task` Class Reference

Represents a task which filters an area of a source `dng_image` to an area of a destination `dng_image`.

```
#include <dng_filter_task.h>
```

Inheritance diagram for `dng_filter_task`:

#### Public Member Functions

- `dng_filter_task` (const `dng_image` &srcImage, `dng_image` &dstImage)
- virtual `dng_rect SrcArea` (const `dng_rect` &dstArea)
- virtual `dng_point SrcTileSize` (const `dng_point` &dstTileSize)
- virtual void `ProcessArea` (uint32 threadIndex, `dng_pixel_buffer` &srcBuffer, `dng_pixel_buffer` &dstBuffer)=0
- virtual void `Start` (uint32 threadCount, const `dng_point` &tileSize, `dng_memory_allocator` \*allocator, `dng_abort_sniffer` \*sniffer)
- virtual void `Process` (uint32 threadIndex, const `dng_rect` &area, `dng_abort_sniffer` \*sniffer)

### Protected Attributes

- const [dng\\_image](#) & **fSrcImage**
- [dng\\_image](#) & **fDstImage**
- uint32 **fSrcPlane**
- uint32 **fSrcPlanes**
- uint32 **fSrcPixelFormat**
- uint32 **fDstPlane**
- uint32 **fDstPlanes**
- uint32 **fDstPixelFormat**
- [dng\\_point](#) **fSrcRepeat**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fSrcBuffer** [[kMaxMPThreads](#)]
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fDstBuffer** [[kMaxMPThreads](#)]

#### 6.20.1 Detailed Description

Represents a task which filters an area of a source [dng\\_image](#) to an area of a destination [dng\\_image](#).

#### 6.20.2 Constructor & Destructor Documentation

##### 6.20.2.1 [dng\\_filter\\_task::dng\\_filter\\_task](#) (const [dng\\_image](#) & *srcImage*, [dng\\_image](#) & *dstImage*)

Construct a filter task given a source and destination images.

#### Parameters:

- srcImage* Image from which source pixels are read.  
*dstImage* Image to which result pixels are written.

#### 6.20.3 Member Function Documentation

##### 6.20.3.1 [void dng\\_filter\\_task::Process](#) (uint32 *threadIndex*, const [dng\\_rect](#) & *area*, [dng\\_abort\\_sniffer](#) \* *sniffer*) [[virtual](#)]

Process one tile or partitioned area. Should not be overridden. Instead, override [ProcessArea](#), which is where to implement filter processing for a specific type of [dng\\_filter\\_task](#). There is no allocator parameter as all allocation should be done in [Start](#).

**Parameters:**

- threadIndex* 0 to threadCount - 1 index indicating which thread this is. (Can be used to get a thread-specific buffer allocated in the Start method.)
- area* Size of tiles to be used for sizing buffers, etc. (Edges of processing can be smaller.)
- sniffer* [dng\\_abort\\_sniffer](#) to use to check for user cancellation and progress updates.

Implements [dng\\_area\\_task](#).

References [dng\\_image::edge\\_repeat](#), [dng\\_image::Get\(\)](#), [ProcessArea\(\)](#), [dng\\_image::Put\(\)](#), and [SrcArea\(\)](#).

**6.20.3.2 virtual void dng\_filter\_task::ProcessArea (uint32 threadIndex, dng\_pixel\_buffer & srcBuffer, dng\_pixel\_buffer & dstBuffer)** [pure virtual]

Implements filtering operation from one buffer to another. Source and destination pixels are set up in member fields of this class. Ideally, no allocation should be done in this routine.

**Parameters:**

- threadIndex* The thread on which this routine is being called, between 0 and threadCount - 1 for the threadCount passed to Start method.
- srcBuffer* Input area and source pixels.
- dstBuffer* Output area and destination pixels.

Referenced by [Process\(\)](#).

**6.20.3.3 virtual dng\_rect dng\_filter\_task::SrcArea (const dng\_rect & dstArea)** [inline, virtual]

Compute the source area needed for a given destination area. Default implementation assumes destination area is equal to source area for all cases.

**Parameters:**

- dstArea* Area to for which pixels will be computed.

**Return values:**

- The* source area needed as input to calculate the requested destination area.

Referenced by Process(), and SrcTileSize().

#### 6.20.3.4 virtual dng\_point dng\_filter\_task::SrcTileSize (const dng\_point & dstTileSize) [inline, virtual]

Given a destination tile size, calculate input tile size. Similar to SrcArea, and should seldom be overridden.

##### Parameters:

*dstTileSize* The destination tile size that is targeted for output.

##### Return values:

*The* source tile size needed to compute a tile of the destination size.

References SrcArea().

Referenced by Start().

#### 6.20.3.5 void dng\_filter\_task::Start (uint32 threadCount, const dng\_point & tileSize, dng\_memory\_allocator \* allocator, dng\_abort\_sniffer \* sniffer) [virtual]

Called prior to processing on specific threads. Can be used to allocate per-thread memory buffers, etc.

##### Parameters:

*threadCount* Total number of threads that will be used for processing. Less than or equal to MaxThreads of [dng\\_area\\_task](#).

*tileSize* Size of source tiles which will be processed. (Not all tiles will be this size due to edge conditions.)

*allocator* [dng\\_memory\\_allocator](#) to use for allocating temporary buffers, etc.

*sniffer* Sniffer to test for user cancellation and to set up progress.

Reimplemented from [dng\\_area\\_task](#).

References dng\_memory\_allocator::Allocate(), and SrcTileSize().

The documentation for this class was generated from the following files:

- [dng\\_filter\\_task.h](#)
- [dng\\_filter\\_task.cpp](#)

## 6.21 dng\_fingerprint Class Reference

Container fingerprint (MD5 only at present).

```
#include <dng_fingerprint.h>
```

### Public Member Functions

- bool [IsNull](#) () const  
*Check if fingerprint is all zeros.*
- bool [IsValid](#) () const  
*Same as IsNull but expresses intention of testing validity.*
- void [Clear](#) ()  
*Set to all zeros, a value used to indicate an invalid fingerprint.*
- bool [operator==](#) (const [dng\\_fingerprint](#) &print) const  
*Test if two fingerprints are equal.*
- bool [operator!=](#) (const [dng\\_fingerprint](#) &print) const  
*Test if two fingerprints are not equal.*
- uint32 [Collapse32](#) () const  
*Produce a 32-bit hash value from fingerprint used for faster hashing of fingerprints.*

### Public Attributes

- uint8 **data** [16]

#### 6.21.1 Detailed Description

Container fingerprint (MD5 only at present).

The documentation for this class was generated from the following files:

- [dng\\_fingerprint.h](#)
- [dng\\_fingerprint.cpp](#)

## 6.22 `dng_function_exposure_ramp` Class Reference

Curve for pre-exposure-compensation adjustment based on noise floor, shadows, and highlight level.

```
#include <dng_render.h>
```

Inheritance diagram for `dng_function_exposure_ramp`:

### Public Member Functions

- `dng_function_exposure_ramp` (real64 white, real64 black, real64 minBlack)
- virtual real64 `Evaluate` (real64 x) const

### Public Attributes

- real64 `fSlope`
- real64 `fBlack`
- real64 `fRadius`
- real64 `fQScale`

#### 6.22.1 Detailed Description

Curve for pre-exposure-compensation adjustment based on noise floor, shadows, and highlight level.

#### 6.22.2 Member Function Documentation

##### 6.22.2.1 `real64 dng_function_exposure_ramp::Evaluate (real64 x) const` [virtual]

Return the mapping for value `x`. This method must be implemented by a derived class of `dng_id_function` and the derived class determines the lookup method and function used.

#### Parameters:

`x` A value between 0.0 and 1.0 (inclusive).

#### Return values:

*Mapped* value for `x`

Implements [dng\\_1d\\_function](#).

The documentation for this class was generated from the following files:

- [dng\\_render.h](#)
- `dng_render.cpp`

## 6.23 `dng_function_exposure_tone` Class Reference

Exposure compensation curve for a given compensation amount in stops using quadric for roll-off.

```
#include <dng_render.h>
```

Inheritance diagram for `dng_function_exposure_tone`:

### Public Member Functions

- `dng_function_exposure_tone` (real64 exposure)
- virtual real64 [Evaluate](#) (real64 x) const  
*Returns output value for a given input tone.*

### Protected Attributes

- bool `fIsNOP`
- real64 `fSlope`
- real64 `a`
- real64 `b`
- real64 `c`

#### 6.23.1 Detailed Description

Exposure compensation curve for a given compensation amount in stops using quadric for roll-off.

The documentation for this class was generated from the following files:

- [dng\\_render.h](#)
- `dng_render.cpp`

## 6.24 `dng_function_gamma_encode` Class Reference

Encoding gamma curve for a given color space.

```
#include <dng_render.h>
```

Inheritance diagram for `dng_function_gamma_encode`:

### Public Member Functions

- `dng_function_gamma_encode` (const [dng\\_color\\_space](#) &space)
- virtual `real64 Evaluate` (`real64 x`) const

### Protected Attributes

- const [dng\\_color\\_space](#) & `fSpace`

#### 6.24.1 Detailed Description

Encoding gamma curve for a given color space.

#### 6.24.2 Member Function Documentation

##### 6.24.2.1 virtual `real64 dng_function_gamma_encode::Evaluate (real64 x) const` [virtual]

Return the mapping for value `x`. This method must be implemented by a derived class of [dng\\_1d\\_function](#) and the derived class determines the lookup method and function used.

#### Parameters:

`x` A value between 0.0 and 1.0 (inclusive).

#### Return values:

*Mapped* value for `x`

Implements [dng\\_1d\\_function](#).

The documentation for this class was generated from the following file:

- [dng\\_render.h](#)

## 6.25 dng\_function\_GammaEncode\_1\_8 Class Reference

A [dng\\_1d\\_function](#) for gamma encoding with 1.8 gamma.

```
#include <dng_color_space.h>
```

Inheritance diagram for `dng_function_GammaEncode_1_8::`

### Public Member Functions

- virtual real64 [Evaluate](#) (real64 x) const
- virtual real64 [EvaluateInverse](#) (real64 y) const

### Static Public Member Functions

- static const [dng\\_1d\\_function](#) & [Get](#) ()

### 6.25.1 Detailed Description

A [dng\\_1d\\_function](#) for gamma encoding with 1.8 gamma.

### 6.25.2 Member Function Documentation

#### 6.25.2.1 real64 `dng_function_GammaEncode_1_8::Evaluate` (real64 x) const [virtual]

Return the mapping for value x. This method must be implemented by a derived class of [dng\\_1d\\_function](#) and the derived class determines the lookup method and function used.

#### Parameters:

- x* A value between 0.0 and 1.0 (inclusive).

#### Return values:

- Mapped* value for x

Implements [dng\\_1d\\_function](#).

### 6.25.2.2 `real64 dng_function_GammaEncode_1_8::EvaluateInverse (real64 y)` `const` [virtual]

Return the reverse mapped value for  $y$ . This method can be implemented by derived classes. The default implementation uses Newton's method to solve for  $x$  such that  $\text{Evaluate}(x) == y$ .

#### Parameters:

- $y$  A value to reverse map. Should be within the range of the function implemented by this [`dng\_1d\_function`](#).

#### Return values:

- A value  $x$  such that  $\text{Evaluate}(x) == y$  (to very close approximation).

Reimplemented from [`dng\_1d\_function`](#).

The documentation for this class was generated from the following files:

- [`dng\_color\_space.h`](#)
- `dng_color_space.cpp`

## 6.26 `dng_function_GammaEncode_2_2` Class Reference

A [`dng\_1d\_function`](#) for gamma encoding with 2.2 gamma.

```
#include <dng_color_space.h>
```

Inheritance diagram for `dng_function_GammaEncode_2_2::`

#### Public Member Functions

- virtual `real64 Evaluate` (`real64 x`) `const`
- virtual `real64 EvaluateInverse` (`real64 y`) `const`

#### Static Public Member Functions

- static `const dng_1d_function & Get` ()

### 6.26.1 Detailed Description

A [`dng\_1d\_function`](#) for gamma encoding with 2.2 gamma.

## 6.26.2 Member Function Documentation

### 6.26.2.1 `real64 dng_function_GammaEncode_2_2::Evaluate (real64 x) const` [virtual]

Return the mapping for value  $x$ . This method must be implemented by a derived class of [dng\\_1d\\_function](#) and the derived class determines the lookup method and function used.

#### Parameters:

$x$  A value between 0.0 and 1.0 (inclusive).

#### Return values:

*Mapped* value for  $x$

Implements [dng\\_1d\\_function](#).

### 6.26.2.2 `real64 dng_function_GammaEncode_2_2::EvaluateInverse (real64 y)` `const` [virtual]

Return the reverse mapped value for  $y$ . This method can be implemented by derived classes. The default implementation uses Newton's method to solve for  $x$  such that  $\text{Evaluate}(x) == y$ .

#### Parameters:

$y$  A value to reverse map. Should be within the range of the function implemented by this [dng\\_1d\\_function](#).

#### Return values:

A value  $x$  such that  $\text{Evaluate}(x) == y$  (to very close approximation).

Reimplemented from [dng\\_1d\\_function](#).

The documentation for this class was generated from the following files:

- [dng\\_color\\_space.h](#)
- [dng\\_color\\_space.cpp](#)

## 6.27 `dng_function_GammaEncode_sRGB` Class Reference

A [`dng\_1d\_function`](#) for gamma encoding in sRGB color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for `dng_function_GammaEncode_sRGB`:

### Public Member Functions

- virtual real64 [Evaluate](#) (real64 x) const
- virtual real64 [EvaluateInverse](#) (real64 y) const

### Static Public Member Functions

- static const [dng\\_1d\\_function](#) & [Get](#) ()

#### 6.27.1 Detailed Description

A [`dng\_1d\_function`](#) for gamma encoding in sRGB color space.

#### 6.27.2 Member Function Documentation

##### 6.27.2.1 `real64 dng_function_GammaEncode_sRGB::Evaluate (real64 x) const` [virtual]

Return the mapping for value x. This method must be implemented by a derived class of [`dng\_1d\_function`](#) and the derived class determines the lookup method and function used.

#### Parameters:

*x* A value between 0.0 and 1.0 (inclusive).

#### Return values:

*Mapped* value for x

Implements [`dng\_1d\_function`](#).

### 6.27.2.2 `real64 dng_function_GammaEncode_sRGB::EvaluateInverse (real64 y) const` [virtual]

Return the reverse mapped value for `y`. This method can be implemented by derived classes. The default implementation uses Newton's method to solve for `x` such that `Evaluate(x) == y`.

#### Parameters:

- `y` A value to reverse map. Should be within the range of the function implemented by this `dng_1d_function`.

#### Return values:

- A value `x` such that `Evaluate(x) == y` (to very close approximation).

Reimplemented from `dng_1d_function`.

The documentation for this class was generated from the following files:

- `dng_color_space.h`
- `dng_color_space.cpp`

## 6.28 `dng_host` Class Reference

The main class for communication between the application and the DNG SDK. Used to customize memory allocation and other behaviors.

```
#include <dng_host.h>
```

#### Public Member Functions

- `dng_host` (`dng_memory_allocator` \*allocator=NULL, `dng_abort_sniffer` \*sniffer=NULL)
- virtual `~dng_host` ()
- `dng_memory_allocator` & `Allocator` ()  
*Getter for host's memory allocator.*
- virtual `dng_memory_block` \* `Allocate` (uint32 logicalSize)
- void `SetSniffer` (`dng_abort_sniffer` \*sniffer)  
*Setter for host's abort sniffer.*
- `dng_abort_sniffer` \* `Sniffer` ()

*Getter for host's abort sniffer.*

- virtual void [SniffForAbort](#) ()
- void [SetNeedsMeta](#) (bool needs)
- bool [NeedsMeta](#) () const

*Getter for flag determining whether all XMP metadata should be parsed.*

- void [SetNeedsImage](#) (bool needs)
- bool [NeedsImage](#) () const

*Setter for flag determining whether DNG image data is needed.*

- void [SetForPreview](#) (bool preview)
- bool [ForPreview](#) () const
- void [SetMinimumSize](#) (uint32 size)
- uint32 [MinimumSize](#) () const

*Getter for the minimum preview size.*

- void [SetPreferredSize](#) (uint32 size)
- uint32 [PreferredSize](#) () const

*Getter for the preferred preview size.*

- void [SetMaximumSize](#) (uint32 size)
- uint32 [MaximumSize](#) () const

*Getter for the maximum preview size.*

- void [SetCropFactor](#) (real64 cropFactor)
- real64 [CropFactor](#) () const

*Getter for the cropping factor.*

- void [ValidateSizes](#) ()

*Makes sures minimum, preferred, and maximum sizes are reasonable.*

- void [SetSaveDNGVersion](#) (uint32 version)
- virtual uint32 [SaveDNGVersion](#) () const

*Getter for what version to save DNG file compatible with.*

- void [SetSaveLinearDNG](#) (bool linear)
- virtual bool [SaveLinearDNG](#) (const [dng\\_negative](#) &negative) const

*Getter for flag determining whether to save a linear DNG file.*

- void [SetKeepOriginalFile](#) (bool keep)
- bool [KeepOriginalFile](#) ()

*Getter for flag determining whether to keep original RAW file data.*

- virtual bool `IsTransientError` (`dng_error_code` code)
- virtual void `PerformAreaTask` (`dng_area_task` &task, const `dng_rect` &area)
- virtual `dng_exif` \* `Make_dng_exif` ()
- virtual `dng_shared` \* `Make_dng_shared` ()
- virtual `dng_ifd` \* `Make_dng_ifd` ()
- virtual `dng_negative` \* `Make_dng_negative` ()
- virtual `dng_image` \* `Make_dng_image` (const `dng_rect` &bounds, uint32 planes, uint32 pixelType)
- virtual `dng_opcode` \* `Make_dng_opcode` (uint32 opcodeID, `dng_stream` &stream)
- virtual void `ApplyOpcodeList` (`dng_opcode_list` &list, `dng_negative` &negative, `AutoPtr`< `dng_image` > &image)

### 6.28.1 Detailed Description

The main class for communication between the application and the DNG SDK. Used to customize memory allocation and other behaviors.

`dng_host` allows setting parameters for the DNG conversion, mediates callback style interactions between the host application and the DNG SDK, and allows controlling certain internal behavior of the SDK such as memory allocation. Many applications will be able to use the default implementation of `dng_host` by just setting the `dng_memory_allocator` and `dng_abort_sniffer` in the constructor. More complex interactions will require deriving a class from `dng_host`.

Multiple `dng_host` objects can be allocated in a single process. This may be useful for DNG processing on separate threads. (Distinct `dng_host` objects are completely thread-safe for read/write. The application is responsible for establishing mutual exclusion for read/write access to a single `dng_host` object if it is used in multiple threads.)

### 6.28.2 Constructor & Destructor Documentation

#### 6.28.2.1 `dng_host::dng_host` (`dng_memory_allocator` \* *allocator* = NULL, `dng_abort_sniffer` \* *sniffer* = NULL)

Allocate a `dng_host` object, possibly with custom allocator and sniffer.

#### Parameters:

*allocator* Allows controlling all memory allocation done via this `dng_host`. Defaults to singleton global `dng_memory_allocator`, which calls `new/delete` `dng_malloc_block` for appropriate size.

*sniffer* Used to periodically check if pending DNG conversions should be aborted and to communicate progress updates. Defaults to singleton global [dng\\_abort\\_sniffer](#), which never aborts and ignores progress updated.

### 6.28.2.2 dng\_host::~~dng\_host () [virtual]

Clean up direct memory for [dng\\_host](#). Memory allocator and abort sniffer are not deleted. Objects such as [dng\\_image](#) and others returned from host can still be used after host is deleted.

## 6.28.3 Member Function Documentation

### 6.28.3.1 dng\_memory\_block \* dng\_host::Allocate (uint32 *logicalSize*) [virtual]

Allocate a new [dng\\_memory\\_block](#) using the host's memory allocator. Uses the [Allocator\(\)](#) property of host to allocate a new block of memory. Will call [ThrowMemoryFull](#) if block cannot be allocated.

#### Parameters:

*logicalSize* Number of usable bytes returned [dng\\_memory\\_block](#) must contain.

References [dng\\_memory\\_allocator::Allocate\(\)](#), and [Allocator\(\)](#).

Referenced by [dng\\_mosaic\\_info::InterpolateGeneric\(\)](#).

### 6.28.3.2 void dng\_host::ApplyOpcodeList (dng\_opcode\_list & *list*, dng\_negative & *negative*, AutoPtr< dng\_image > & *image*) [virtual]

Factory method to apply a [dng\\_opcode\\_list](#). Can be used to override opcode list applications.

### 6.28.3.3 bool dng\_host::ForPreview () const [inline]

Getter for flag determining whether image should be preview quality. Preview quality images may be rendered more quickly. Current DNG SDK does not change rendering

behavior based on this flag, but derived versions may use this getter to choose between a slower more accurate path and a faster "good enough for preview" one. Data produce with `ForPreview` set to true should not be written back to a DNG file, except as a preview image.

#### 6.28.3.4 `bool dng_host::IsTransientError (dng_error_code code)` [virtual]

Determine if an error is the result of a temporary, but planned-for occurrence such as user cancellation or memory exhaustion. This method is sometimes used to determine whether to try and continue processing a DNG file despite errors in the file format, etc. In such cases, processing will be continued if `IsTransientError` returns false. This is so that user cancellation and memory exhaustion always terminate processing.

##### Parameters:

*code* Error to test for transience.

#### 6.28.3.5 `dng_exif * dng_host::Make_dng_exif ()` [virtual]

Factory method for `dng_exif` class. Can be used to customize allocation or to ensure a derived class is used instead of `dng_exif`.

References `ThrowMemoryFull()`.

Referenced by `dng_info::Parse()`.

#### 6.28.3.6 `dng_ifd * dng_host::Make_dng_ifd ()` [virtual]

Factory method for `dng_ifd` class. Can be used to customize allocation or to ensure a derived class is used instead of `dng_ifd`.

References `ThrowMemoryFull()`.

Referenced by `dng_info::Parse()`.

#### 6.28.3.7 `dng_image * dng_host::Make_dng_image (const dng_rect & bounds, uint32 planes, uint32 pixelType)` [virtual]

Factory method for `dng_image` class. Can be used to customize allocation or to ensure a derived class is used instead of `dng_simple_image`.

References `Allocator()`, and `ThrowMemoryFull()`.

Referenced by `dng_render::Render()`.

#### 6.28.3.8 `dng_negative * dng_host::Make_dng_negative ()` [virtual]

Factory method for `dng_negative` class. Can be used to customize allocation or to ensure a derived class is used instead of `dng_negative`.

References `Allocator()`.

#### 6.28.3.9 `dng_opcode * dng_host::Make_dng_opcode (uint32 opcodeID, dng_stream & stream)` [virtual]

Factory method for parsing `dng_opcode` based classes. Can be used to override opcode implementations.

References `ThrowMemoryFull()`.

#### 6.28.3.10 `dng_shared * dng_host::Make_dng_shared ()` [virtual]

Factory method for `dng_shared` class. Can be used to customize allocation or to ensure a derived class is used instead of `dng_shared`.

References `ThrowMemoryFull()`.

Referenced by `dng_info::Parse()`.

#### 6.28.3.11 `void dng_host::PerformAreaTask (dng_area_task & task, const dng_rect & area)` [virtual]

General top-level bottleneck for image processing tasks. Default implementation calls `dng_area_task::PerformAreaTask` method on task. Can be overridden in derived classes to support multiprocessing, for example.

##### Parameters:

*task* Image processing task to perform on area.

*area* Rectangle over which to perform image processing task.

References Allocator(), dng\_area\_task::Perform(), and Sniffer().

Referenced by dng\_mosaic\_info::InterpolateFast(), dng\_linearization\_info::Linearize(), and dng\_render::Render().

#### 6.28.3.12 void dng\_host::SetCropFactor (real64 *cropFactor*) [inline]

Setter for the cropping factor.

##### Parameters:

*cropFactor* Fraction of image to be used after crop.

#### 6.28.3.13 void dng\_host::SetForPreview (bool *preview*) [inline]

Setter for flag determining whether image should be preview quality, or full quality.

##### Parameters:

*preview* If true, rendered images are for preview.

#### 6.28.3.14 void dng\_host::SetKeepOriginalFile (bool *keep*) [inline]

Setter for flag determining whether to keep original RAW file data.

##### Parameters:

*keep* If true, original RAW data will be kept.

#### 6.28.3.15 void dng\_host::SetMaximumSize (uint32 *size*) [inline]

Setter for the maximum preview size.

##### Parameters:

*size* Maximum pixel size (long side of image).

**6.28.3.16 void dng\_host::SetMinimumSize (uint32 *size*)** [inline]

Setter for the minimum preview size.

**Parameters:**

*size* Minimum pixel size (long side of image).

Referenced by ValidateSizes().

**6.28.3.17 void dng\_host::SetNeedsImage (bool *needs*)** [inline]

Setter for flag determining whether DNG image data is needed. Defaults to true. Image data might not be needed for applications which only manipulate metadata.

**Parameters:**

*needs* If true, image data is needed.

**6.28.3.18 void dng\_host::SetNeedsMeta (bool *needs*)** [inline]

Setter for flag determining whether all XMP metadata should be parsed. Defaults to true. One might not want metadata when doing a quick check to see if a file is readable.

**Parameters:**

*needs* If true, metadata is needed.

**6.28.3.19 void dng\_host::SetPreferredSize (uint32 *size*)** [inline]

Setter for the preferred preview size.

**Parameters:**

*size* Preferred pixel size (long side of image).

Referenced by ValidateSizes().

**6.28.3.20 void dng\_host::SetSaveDNGVersion (uint32 *version*)** [inline]

Setter for what version to save DNG file compatible with.

**Parameters:**

*version* What version to save DNG file compatible with.

**6.28.3.21 void dng\_host::SetSaveLinearDNG (bool *linear*)** [inline]

Setter for flag determining whether to force saving a linear DNG file.

**Parameters:**

*linear* If true, we should force saving a linear DNG file.

**6.28.3.22 void dng\_host::SniffForAbort ()** [virtual]

Check for pending abort. Should call ThrowUserCanceled if an abort is pending.

References Sniffer().

Referenced by dng\_mosaic\_info::InterpolateGeneric().

The documentation for this class was generated from the following files:

- [dng\\_host.h](#)
- [dng\\_host.cpp](#)

**6.29 dng\_ifd Class Reference**

Container for a single image file directory of a digital negative.

```
#include <dng_ifd.h>
```

**Public Types**

- enum { **kMaxTileInfo** = 32 }

### Public Member Functions

- virtual bool **ParseTag** ([dng\\_stream](#) &stream, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset)
- virtual void **PostParse** ()
- virtual bool **IsValidDNG** ([dng\\_shared](#) &shared, uint32 parentCode)
- [dng\\_rect](#) **Bounds** () const
- uint32 **TilesAcross** () const
- uint32 **TilesDown** () const
- uint32 **TilesPerImage** () const
- [dng\\_rect](#) **TileArea** (uint32 rowIndex, uint32 colIndex) const
- virtual uint32 **TileByteCount** (const [dng\\_rect](#) &tile) const
- void **SetSingleStrip** ()
- void **FindTileSize** (uint32 bytesPerTile=128 \*1024, uint32 cellH=16, uint32 cellV=16)
- void **FindStripSize** (uint32 bytesPerStrip=128 \*1024, uint32 cellV=16)
- virtual uint32 **PixelFormat** () const
- virtual bool **IsBaselineJPEG** () const
- virtual bool **CanRead** () const
- virtual void **ReadImage** ([dng\\_host](#) &host, [dng\\_stream](#) &stream, [dng\\_image](#) &image) const

### Public Attributes

- bool **fUsesNewSubFileType**
- uint32 **fNewSubFileType**
- uint32 **fImageWidth**
- uint32 **fImageLength**
- uint32 **fBitsPerSample** [[kMaxSamplesPerPixel](#)]
- uint32 **fCompression**
- uint32 **fPredictor**
- uint32 **fPhotometricInterpretation**
- uint32 **fFillOrder**
- uint32 **fOrientation**
- uint32 **fOrientationType**
- uint64 **fOrientationOffset**
- bool **fOrientationBigEndian**
- uint32 **fSamplesPerPixel**
- uint32 **fPlanarConfiguration**
- real64 **fXResolution**
- real64 **fYResolution**
- uint32 **fResolutionUnit**
- bool **fUsesStrips**

- bool **fUsesTiles**
- uint32 **fTileWidth**
- uint32 **fTileLength**
- uint32 **fTileOffsetsType**
- uint32 **fTileOffsetsCount**
- uint64 **fTileOffsetsOffset**
- uint64 **fTileOffset** [kMaxTileInfo]
- uint32 **fTileByteCountsType**
- uint32 **fTileByteCountsCount**
- uint64 **fTileByteCountsOffset**
- uint32 **fTileByteCount** [kMaxTileInfo]
- uint32 **fSubIFDsCount**
- uint64 **fSubIFDsOffset**
- uint32 **fExtraSamplesCount**
- uint32 **fExtraSamples** [kMaxSamplesPerPixel]
- uint32 **fSampleFormat** [kMaxSamplesPerPixel]
- uint32 **fJPEGTablesCount**
- uint64 **fJPEGTablesOffset**
- uint64 **fJPEGInterchangeFormat**
- uint32 **fJPEGInterchangeFormatLength**
- real64 **fYCbCrCoefficientR**
- real64 **fYCbCrCoefficientG**
- real64 **fYCbCrCoefficientB**
- uint32 **fYCbCrSubSampleH**
- uint32 **fYCbCrSubSampleV**
- uint32 **fYCbCrPositioning**
- real64 **fReferenceBlackWhite** [6]
- uint32 **fCFARepeatPatternRows**
- uint32 **fCFARepeatPatternCols**
- uint8 **fCFAPattern** [kMaxCFAPattern][kMaxCFAPattern]
- uint8 **fCFAPlaneColor** [kMaxColorPlanes]
- uint32 **fCFALayout**
- uint32 **fLinearizationTableType**
- uint32 **fLinearizationTableCount**
- uint64 **fLinearizationTableOffset**
- uint32 **fBlackLevelRepeatRows**
- uint32 **fBlackLevelRepeatCols**
- real64 **fBlackLevel** [kMaxBlackPattern][kMaxBlackPattern][kMaxSamplesPerPixel]
  
- uint32 **fBlackLevelDeltaHType**
- uint32 **fBlackLevelDeltaHCount**
- uint64 **fBlackLevelDeltaHOffset**
- uint32 **fBlackLevelDeltaVType**

- uint32 **fBlackLevelDeltaVCount**
- uint64 **fBlackLevelDeltaVOffset**
- real64 **fWhiteLevel** [[kMaxSamplesPerPixel](#)]
- dng\_urational **fDefaultScaleH**
- dng\_urational **fDefaultScaleV**
- dng\_urational **fBestQualityScale**
- dng\_urational **fDefaultCropOriginH**
- dng\_urational **fDefaultCropOriginV**
- dng\_urational **fDefaultCropSizeH**
- dng\_urational **fDefaultCropSizeV**
- uint32 **fBayerGreenSplit**
- dng\_urational **fChromaBlurRadius**
- dng\_urational **fAntiAliasStrength**
- dng\_rect **fActiveArea**
- uint32 **fMaskedAreaCount**
- dng\_rect **fMaskedArea** [[kMaxMaskedAreas](#)]
- uint32 **fRowInterleaveFactor**
- uint32 **fSubTileBlockRows**
- uint32 **fSubTileBlockCols**
- dng\_preview\_info **fPreviewInfo**
- uint32 **fOpcodeList1Count**
- uint64 **fOpcodeList1Offset**
- uint32 **fOpcodeList2Count**
- uint64 **fOpcodeList2Offset**
- uint32 **fOpcodeList3Count**
- uint64 **fOpcodeList3Offset**
- bool **fLosslessJPEGBug16**
- uint32 **fSampleBitShift**
- uint64 **fThisIFD**
- uint64 **fNextIFD**

### Protected Member Functions

- virtual bool **IsValidCFA** (dng\_shared &shared, uint32 parentCode)

#### 6.29.1 Detailed Description

Container for a single image file directory of a digital negative.

See [DNG 1.1.0 specification](#) for documentation of specific tags.

The documentation for this class was generated from the following files:

- [dng\\_ifd.h](#)
- [dng\\_ifd.cpp](#)

## 6.30 `dng_image` Class Reference

Base class for holding image data in DNG SDK. See [dng\\_simple\\_image](#) for derived class most often used in DNG SDK.

```
#include <dng_image.h>
```

Inheritance diagram for `dng_image::`

### Public Types

- enum [edge\\_option](#) { [edge\\_none](#), [edge\\_zero](#), [edge\\_repeat](#), [edge\\_repeat\\_zero\\_last](#) }

*How to handle requests to get image areas outside the image bounds.*

### Public Member Functions

- virtual [dng\\_image](#) \* **Clone** () const
- const [dng\\_rect](#) & **Bounds** () const  
*Getter method for bounds of an image.*
- [dng\\_point](#) **Size** () const  
*Getter method for size of an image.*
- uint32 **Width** () const  
*Getter method for width of an image.*
- uint32 **Height** () const  
*Getter method for height of an image.*
- uint32 **Planes** () const  
*Getter method for number of planes in an image.*
- uint32 **PixelType** () const
- virtual void **SetPixelType** (uint32 pixelType)
- uint32 **PixelSize** () const
- uint32 **PixelRange** () const
- virtual [dng\\_rect](#) **RepeatingTile** () const  
*Getter for best "tile stride" for accessing image.*

- void `Get` (`dng_pixel_buffer` &buffer, `edge_option` edgeOption=edge\_none, uint32 repeatV=1, uint32 repeatH=1) const
- void `Put` (const `dng_pixel_buffer` &buffer)
- virtual void `Trim` (const `dng_rect` &r)
- virtual void `Rotate` (const `dng_orientation` &orientation)
- void `CopyArea` (const `dng_image` &src, const `dng_rect` &area, uint32 srcPlane, uint32 dstPlane, uint32 planes)
- void `CopyArea` (const `dng_image` &src, const `dng_rect` &area, uint32 plane, uint32 planes)
- bool `EqualArea` (const `dng_image` &rhs, const `dng_rect` &area, uint32 plane, uint32 planes) const
- void `SetConstant_uint8` (uint8 value, const `dng_rect` &area)
- void `SetConstant_uint8` (uint8 value)
- void `SetConstant_uint16` (uint16 value, const `dng_rect` &area)
- void `SetConstant_uint16` (uint16 value)
- void `SetConstant_int16` (int16 value, const `dng_rect` &area)
- void `SetConstant_int16` (int16 value)
- void `SetConstant_uint32` (uint32 value, const `dng_rect` &area)
- void `SetConstant_uint32` (uint32 value)
- void `SetConstant_real32` (real32 value, const `dng_rect` &area)
- void `SetConstant_real32` (real32 value)
- virtual void `GetRepeat` (`dng_pixel_buffer` &buffer, const `dng_rect` &srcArea, const `dng_rect` &dstArea) const

### Protected Member Functions

- `dng_image` (const `dng_rect` &bounds, uint32 planes, uint32 pixelType)
- virtual void `AcquireTileBuffer` (`dng_tile_buffer` &buffer, const `dng_rect` &area, bool dirty) const
- virtual void `ReleaseTileBuffer` (`dng_tile_buffer` &buffer) const
- virtual void `DoGet` (`dng_pixel_buffer` &buffer) const
- virtual void `DoPut` (const `dng_pixel_buffer` &buffer)
- void `GetEdge` (`dng_pixel_buffer` &buffer, `edge_option` edgeOption, const `dng_rect` &srcArea, const `dng_rect` &dstArea) const
- virtual void `SetConstant` (uint32 value, const `dng_rect` &area)

### Protected Attributes

- `dng_rect` `fBounds`
- uint32 `fPlanes`
- uint32 `fPixelType`

## Friends

- class [dng\\_tile\\_buffer](#)

### 6.30.1 Detailed Description

Base class for holding image data in DNG SDK. See [dng\\_simple\\_image](#) for derived class most often used in DNG SDK.

### 6.30.2 Member Enumeration Documentation

#### 6.30.2.1 `enum dng_image::edge_option`

How to handle requests to get image areas outside the image bounds.

#### Enumerator:

*edge\_none* Leave edge pixels unchanged.

*edge\_zero* Pad with zeros.

*edge\_repeat* Repeat edge pixels.

*edge\_repeat\_zero\_last* Repeat edge pixels, except for last plane which is zero padded.

### 6.30.3 Member Function Documentation

#### 6.30.3.1 `void dng_image::CopyArea (const dng_image & src, const dng_rect & area, uint32 plane, uint32 planes) [inline]`

Copy image data from an area of one image to same area of another.

#### Parameters:

*src* Image to copy from.

*area* Rectangle of images to copy.

*plane* Plane to start copying in src and this.

*planes* Number of planes to copy.

References `CopyArea()`.

### 6.30.3.2 void dng\_image::CopyArea (const dng\_image & *src*, const dng\_rect & *area*, uint32 *srcPlane*, uint32 *dstPlane*, uint32 *planes*)

Copy image data from an area of one image to same area of another.

#### Parameters:

- src* Image to copy from.
- area* Rectangle of images to copy.
- srcPlane* Plane to start copying in src.
- dstPlane* Plane to start copying in this.
- planes* Number of planes to copy.

References dng\_pixel\_buffer::CopyArea().

Referenced by CopyArea().

### 6.30.3.3 bool dng\_image::EqualArea (const dng\_image & *rhs*, const dng\_rect & *area*, uint32 *plane*, uint32 *planes*) const

Return true if the contents of an area of the image are the same as those of another.

#### Parameters:

- rhs* Image to compare against.
- area* Rectangle of image to test.
- plane* Plane to start comparing.
- planes* Number of planes to compare.

References dng\_pixel\_buffer::EqualArea().

### 6.30.3.4 void dng\_image::Get (dng\_pixel\_buffer & *buffer*, edge\_option *edgeOption* = edge\_none, uint32 *repeatV* = 1, uint32 *repeatH* = 1) const

Get a pixel buffer of data on image with proper edge padding.

#### Parameters:

- buffer* Receives resulting pixel buffer.

*edgeOption* edge\_option describing how to pad edges.

*repeatV* Amount of repeated padding needed in vertical for edge\_repeat and edge\_repeat\_zero\_last edgeOption cases.

*repeatH* Amount of repeated padding needed in horizontal for edge\_repeat and edge\_repeat\_zero\_last edgeOption cases.

References dng\_pixel\_buffer::DirtyPixel(), and edge\_none.

Referenced by dng\_mosaic\_info::InterpolateGeneric(), and dng\_filter\_task::Process().

#### 6.30.3.5 uint32 dng\_image::PixelRange () const

Getter for pixel range. For unsigned types, range is 0 to return value. For signed types, range is return value - 0x8000U. For ttFloat type, pixel range is 0.0 to 1.0 and this routine returns 1.

#### 6.30.3.6 uint32 dng\_image::PixelSize () const

Getter for pixel size.

##### Return values:

*Size, in* bytes, of pixel type for this image .

References PixelType().

Referenced by dng\_mosaic\_info::InterpolateGeneric(), and SetPixelType().

#### 6.30.3.7 uint32 dng\_image::PixelType () const [inline]

Getter for pixel type.

##### Return values:

*See* dng\_tagtypes.h . Valid values are ttByte, ttShort, ttShort, ttLong, ttFloat .

Referenced by dng\_mosaic\_info::InterpolateGeneric(), PixelSize(), dng\_render::Render(), and dng\_image\_writer::WriteTIFFWithProfile().

**6.30.3.8 void dng\_image::Put (const dng\_pixel\_buffer & *buffer*)**

Put a pixel buffer into image.

**Parameters:**

*buffer* Pixel buffer to copy from.

References dng\_pixel\_buffer::ConstPixel(), and Planes().

Referenced by dng\_mosaic\_info::InterpolateGeneric(), and dng\_filter\_task::Process().

**6.30.3.9 void dng\_image::Rotate (const dng\_orientation & *orientation*)**  
[virtual]

Rotate image to reflect given orientation change.

**Parameters:**

*orientation* Directive to rotate image in a certain way.

Reimplemented in [dng\\_simple\\_image](#).

References ThrowProgramError().

**6.30.3.10 void dng\_image::SetPixelType (uint32 *pixelType*)** [virtual]

Setter for pixel type.

**Parameters:**

*pixelType* The new pixel type .

Reimplemented in [dng\\_simple\\_image](#).

References PixelSize(), and ThrowProgramError().

**6.30.3.11 void dng\_image::Trim (const dng\_rect & *r*)** [virtual]

Shrink bounds of image to given rectangle.

**Parameters:**

- r* Rectangle to crop to.

Reimplemented in [dng\\_simple\\_image](#).

References [Bounds\(\)](#), and [ThrowProgramError\(\)](#).

The documentation for this class was generated from the following files:

- [dng\\_image.h](#)
- [dng\\_image.cpp](#)

**6.31 `dng_image_writer` Class Reference**

Support for writing [dng\\_image](#) or [dng\\_negative](#) instances to a [dng\\_stream](#) in TIFF or DNG format.

```
#include <dng_image_writer.h>
```

**Public Member Functions**

- virtual void **WriteImage** ([dng\\_host](#) &host, const [dng\\_ifd](#) &ifd, [dng\\_basic\\_tag\\_set](#) &basic, [dng\\_stream](#) &stream, const [dng\\_image](#) &image, uint32 fakeChannels=1)
- virtual void **WriteTIFF** ([dng\\_host](#) &host, [dng\\_stream](#) &stream, const [dng\\_image](#) &image, uint32 photometricInterpretation=piBlackIsZero, uint32 compression=ccUncompressed, [dng\\_negative](#) \*negative=NULL, const [dng\\_color\\_space](#) \*space=NULL, const [dng\\_resolution](#) \*resolution=NULL, const [dng\\_jpeg\\_preview](#) \*thumbnail=NULL, const [dng\\_memory\\_block](#) \*imageResources=NULL)
- virtual void **WriteTIFFWithProfile** ([dng\\_host](#) &host, [dng\\_stream](#) &stream, const [dng\\_image](#) &image, uint32 photometricInterpretation=piBlackIsZero, uint32 compression=ccUncompressed, [dng\\_negative](#) \*negative=NULL, const void \*profileData=NULL, uint32 profileSize=0, const [dng\\_resolution](#) \*resolution=NULL, const [dng\\_jpeg\\_preview](#) \*thumbnail=NULL, const [dng\\_memory\\_block](#) \*imageResources=NULL)
- virtual void **WriteDNG** ([dng\\_host](#) &host, [dng\\_stream](#) &stream, const [dng\\_negative](#) &negative, const [dng\\_image\\_preview](#) &thumbnail, uint32 compression=ccJPEG, const [dng\\_preview\\_list](#) \*previewList=NULL)

**Protected Types**

- enum { **kImageBufferSize** = 128 \* 1024 }

### Protected Member Functions

- virtual uint32 **CompressedBufferSize** (const [dng\\_ifd](#) &ifd, uint32 uncompressedSize)
- virtual void **EncodePredictor** ([dng\\_host](#) &host, const [dng\\_ifd](#) &ifd, [dng\\_pixel\\_buffer](#) &buffer)
- virtual void **ByteSwapBuffer** ([dng\\_host](#) &host, [dng\\_pixel\\_buffer](#) &buffer)
- void **ReorderSubTileBlocks** (const [dng\\_ifd](#) &ifd, [dng\\_pixel\\_buffer](#) &buffer)
- virtual void **WriteData** ([dng\\_host](#) &host, const [dng\\_ifd](#) &ifd, [dng\\_stream](#) &stream, [dng\\_pixel\\_buffer](#) &buffer)
- virtual void **WriteTile** ([dng\\_host](#) &host, const [dng\\_ifd](#) &ifd, [dng\\_stream](#) &stream, const [dng\\_image](#) &image, const [dng\\_rect](#) &tileArea, uint32 fakeChannels)

### Protected Attributes

- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fCompressedBuffer**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fUncompressedBuffer**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fSubTileBlockBuffer**

#### 6.31.1 Detailed Description

Support for writing [dng\\_image](#) or [dng\\_negative](#) instances to a [dng\\_stream](#) in TIFF or DNG format.

#### 6.31.2 Member Function Documentation

**6.31.2.1** void [dng\\_image\\_writer::WriteDNG](#) ([dng\\_host](#) & *host*, [dng\\_stream](#) & *stream*, const [dng\\_negative](#) & *negative*, const [dng\\_image\\_preview](#) & *thumbnail*, uint32 *compression* = ccJPEG, const [dng\\_preview\\_list](#) \* *previewList* = NULL) [virtual]

Write a [dng\\_image](#) to a [dng\\_stream](#) in DNG format.

#### Parameters:

*host* Host interface used for progress updates, abort testing, buffer allocation, etc.

*stream* The [dng\\_stream](#) on which to write the TIFF.

*negative* The image data and metadata (EXIF, IPTC, XMP) to be written.

*thumbnail* Thumbnail image. Must be provided.

*compression* Either ccUncompressed or ccJPEG for lossless JPEG.

*previewList* List of previews (not counting thumbnail) to write to the file. Defaults to empty.

References `dng_negative::AntiAliasStrength()`, `dng_negative::BaselineExposureR()`, `dng_negative::BaselineNoiseR()`, `dng_negative::BaselineSharpnessR()`, `dng_negative::BestQualityScale()`, `dng_stream::BigEndian()`, `dng_memory_data::Buffer_uint32()`, `dng_negative::ChromaBlurRadius()`, `dng_negative::DefaultCropOriginH()`, `dng_negative::DefaultCropOriginV()`, `dng_negative::DefaultCropSizeH()`, `dng_negative::DefaultCropSizeV()`, `dng_negative::DefaultScaleH()`, `dng_negative::DefaultScaleV()`, `dng_mosaic_info::fCFALayout`, `dng_mosaic_info::fCFAPatternSize`, `dng_linearization_info::fLinearizationTable`, `dng_stream::Flush()`, `AutoPtr< T >::Get()`, `dng_mosaic_info::IsColorFilterArray()`, `dng_negative::IsMonochrome()`, `dng_fingerprint::IsValid()`, `kMaxDNGPreviews`, `dng_stream::Length()`, `dng_negative::LocalName()`, `dng_negative::ModelName()`, `dng_negative::NoiseProfile()`, `dng_negative::NoiseReductionApplied()`, `dng_negative::Orientation()`, `dng_stream::Position()`, `dng_stream::Put_uint16()`, `dng_stream::Put_uint32()`, `AutoPtr< T >::Reset()`, `dng_stream::SetLength()`, `dng_stream::SetWritePosition()`, `dng_negative::ShadowScaleR()`, `ThrowImageTooBigDNG()`, and `ThrowProgramError()`.

**6.31.2.2** `void dng_image_writer::WriteTIFF (dng_host & host, dng_stream & stream, const dng_image & image, uint32 photometricInterpretation = piBlackIsZero, uint32 compression = ccUncompressed, dng_negative * negative = NULL, const dng_color_space * space = NULL, const dng_resolution * resolution = NULL, const dng_jpeg_preview * thumbnail = NULL, const dng_memory_block * imageResources = NULL) [virtual]`

Write a `dng_image` to a `dng_stream` in TIFF format.

#### Parameters:

- host* Host interface used for progress updates, abort testing, buffer allocation, etc.
- stream* The `dng_stream` on which to write the TIFF.
- image* The actual image data to be written.
- photometricInterpretation* Either `piBlackIsZero` for monochrome or `piRGB` for RGB images.
- compression* Must be `ccUncompressed`.
- negative* If non-NULL, EXIF, IPTC, and XMP metadata from this negative is written to TIFF.
- space* If non-null and color space has an ICC profile, TIFF will be tagged with this profile. No color space conversion of image data occurs.

*resolution* If non-NULL, TIFF will be tagged with this resolution.

*thumbnail* If non-NULL, will be stored in TIFF as preview image.

*imageResources* If non-NULL, will image resources be stored in TIFF as well.

References `dng_color_space::ICCProfile()`, and `WriteTIFFWithProfile()`.

**6.31.2.3** `void dng_image_writer::WriteTIFFWithProfile (dng_host & host, dng_stream & stream, const dng_image & image, uint32 photometricInterpretation = piBlackIsZero, uint32 compression = ccUncompressed, dng_negative * negative = NULL, const void * profileData = NULL, uint32 profileSize = 0, const dng_resolution * resolution = NULL, const dng_jpeg_preview * thumbnail = NULL, const dng_memory_block * imageResources = NULL) [virtual]`

Write a `dng_image` to a `dng_stream` in TIFF format.

#### Parameters:

*host* Host interface used for progress updates, abort testing, buffer allocation, etc.

*stream* The `dng_stream` on which to write the TIFF.

*image* The actual image data to be written.

*photometricInterpretation* Either `piBlackIsZero` for monochrome or `piRGB` for RGB images.

*compression* Must be `ccUncompressed`.

*negative* If non-NULL, EXIF, IPTC, and XMP metadata from this negative is written to TIFF.

*profileData* If non-null, TIFF will be tagged with this profile. No color space conversion of image data occurs.

*profileSize* The size for the profile data.

*resolution* If non-NULL, TIFF will be tagged with this resolution.

*thumbnail* If non-NULL, will be stored in TIFF as preview image.

*imageResources* If non-NULL, will image resources be stored in TIFF as well.

References `dng_stream::BigEndian()`, `dng_image::Bounds()`, `dng_stream::Flush()`, `AutoPtr< T >::Get()`, `dng_stream::Length()`, `dng_image::PixelType()`, `dng_image::Planes()`, `dng_stream::Position()`, `dng_stream::Put_uint16()`, `dng_stream::Put_uint32()`, `AutoPtr< T >::Reset()`, `dng_stream::SetLength()`, `dng_stream::SetWritePosition()`, `dng_image::Size()`, and `ThrowImageTooBigTIFF()`.

Referenced by `WriteTIFF()`.

The documentation for this class was generated from the following files:

- [dng\\_image\\_writer.h](#)
- [dng\\_image\\_writer.cpp](#)

## 6.32 `dng_info` Class Reference

Top-level structure of DNG file with access to metadata.

```
#include <dng_info.h>
```

### Public Member Functions

- virtual void [Parse](#) ([dng\\_host](#) &host, [dng\\_stream](#) &stream)
- virtual void [PostParse](#) ([dng\\_host](#) &host)  
*Must be called immediately after a successful Parse operation.*
- virtual bool [IsValidDNG](#) ()

### Public Attributes

- uint64 [fTIFFBlockOffset](#)
- uint64 [fTIFFBlockOriginalOffset](#)
- bool [fBigEndian](#)
- uint32 [fMagic](#)
- [AutoPtr](#)< [dng\\_exif](#) > [fExif](#)
- [AutoPtr](#)< [dng\\_shared](#) > [fShared](#)
- int32 [fMainIndex](#)
- uint32 [fIFDCount](#)
- [AutoPtr](#)< [dng\\_ifd](#) > [fIFD](#) [[kMaxSubIFDs](#)+1]
- uint32 [fChainedIFDCount](#)
- [AutoPtr](#)< [dng\\_ifd](#) > [fChainedIFD](#) [[kMaxChainedIFDs](#)]

### Protected Member Functions

- virtual void [ValidateMagic](#) ()
- virtual void [ParseTag](#) ([dng\\_host](#) &host, [dng\\_stream](#) &stream, [dng\\_exif](#) \*exif, [dng\\_shared](#) \*shared, [dng\\_ifd](#) \*ifd, uint32 parentCode, uint32 tagCode, uint32 tagType, uint32 tagCount, uint64 tagOffset, int64 offsetDelta)
- virtual bool [ValidateIFD](#) ([dng\\_stream](#) &stream, uint64 ifdOffset, int64 offsetDelta)
- virtual void [ParseIFD](#) ([dng\\_host](#) &host, [dng\\_stream](#) &stream, [dng\\_exif](#) \*exif, [dng\\_shared](#) \*shared, [dng\\_ifd](#) \*ifd, uint64 ifdOffset, int64 offsetDelta, uint32 parentCode)

- virtual bool **ParseMakerNoteIFD** (`dng_host` &host, `dng_stream` &stream, uint64 ifdSize, uint64 ifdOffset, int64 offsetDelta, uint64 minOffset, uint64 maxOffset, uint32 parentCode)
- virtual void **ParseMakerNote** (`dng_host` &host, `dng_stream` &stream, uint32 makerNoteCount, uint64 makerNoteOffset, int64 offsetDelta, uint64 minOffset, uint64 maxOffset)
- virtual void **ParseSonyPrivateData** (`dng_host` &host, `dng_stream` &stream, uint32 count, uint64 oldOffset, uint64 newOffset)
- virtual void **ParseDNGPrivateData** (`dng_host` &host, `dng_stream` &stream)

### Protected Attributes

- uint32 `fMakerNoteNextIFD`

### 6.32.1 Detailed Description

Top-level structure of DNG file with access to metadata.

See [DNG 1.1.0 specification](#) for information on member fields of this class.

### 6.32.2 Member Function Documentation

#### 6.32.2.1 `bool dng_info::IsValidDNG ()` [virtual]

Test validity of DNG data.

#### Return values:

*true* if stream provided a valid DNG.

References `AutoPtr< T >::Get()`.

#### 6.32.2.2 `void dng_info::Parse (dng_host & host, dng_stream & stream)` [virtual]

Read `dng_info` from a `dng_stream`

#### Parameters:

*host* DNG host used for progress updating, abort testing, buffer allocation, etc.

*stream* Stream to read DNG data from.

References `AutoPtr< T >::Get()`, `dng_stream::Get_uint16()`, `dng_stream::Get_uint32()`, `dng_camera_profile::IsValid()`, `kMaxChainedIFDs`, `kMaxSubIFDs`, `dng_stream::Length()`, `dng_host::Make_dng_exif()`, `dng_host::Make_dng_ifd()`, `dng_host::Make_dng_shared()`, `dng_camera_profile::Parse()`, `dng_stream::Position()`, `dng_stream::PositionInOriginalFile()`, `AutoPtr< T >::Reset()`, `dng_stream::SetBigEndian()`, `dng_stream::SetLittleEndian()`, `dng_stream::SetReadPosition()`, and `ThrowBadFormat()`.

The documentation for this class was generated from the following files:

- [dng\\_info.h](#)
- [dng\\_info.cpp](#)

## 6.33 dng\_iptc Class Reference

Class for reading and holding IPTC metadata associated with a DNG file.

```
#include <dng_iptc.h>
```

### Public Member Functions

- `bool IsEmpty () const`
- `bool NotEmpty () const`
- `void Parse (const void *blockData, uint32 blockSize, uint64 offsetInOriginalFile)`
- `dng_memory_block * Spool (dng_memory_allocator &allocator, bool padForTIFF)`

### Public Attributes

- `bool fForceUTF8`
- `dng_string fTitle`
- `int32 fUrgency`
- `dng_string fCategory`
- `dng_string_list fSupplementalCategories`
- `dng_string_list fKeywords`
- `dng_string fInstructions`
- `dng_date_time_info fDateTimeCreated`
- `dng_string fAuthor`
- `dng_string fAuthorsPosition`
- `dng_string fCity`
- `dng_string fState`
- `dng_string fCountry`

- dng\_string **fCountryCode**
- dng\_string **fLocation**
- dng\_string **fTransmissionReference**
- dng\_string **fHeadline**
- dng\_string **fCredit**
- dng\_string **fSource**
- dng\_string **fCopyrightNotice**
- dng\_string **fDescription**
- dng\_string **fDescriptionWriter**

### Protected Types

- enum **DataSet** {  
    **kRecordVersionSet** = 0, **kObjectNameSet** = 5, **kUrgencySet** = 10, **kCategorySet** = 15,  
    **kSupplementalCategoriesSet** = 20, **kKeywordsSet** = 25, **kSpecialInstructionsSet** = 40, **kDateCreatedSet** = 55,  
    **kTimeCreatedSet** = 60, **kBylineSet** = 80, **kBylineTitleSet** = 85, **kCitySet** = 90,  
    **kSublocationSet** = 92, **kProvinceStateSet** = 95, **kCountryCodeSet** = 100, **kCountryNameSet** = 101,  
    **kOriginalTransmissionReferenceSet** = 103, **kHeadlineSet** = 105, **kCreditSet** = 110, **kSourceSet** = 115,  
    **kCopyrightNoticeSet** = 116, **kCaptionSet** = 120, **kCaptionWriterSet** = 122 }  
• enum **CharSet** { **kCharSetUnknown** = 0, **kCharSetUTF8** = 1 }

### Protected Member Functions

- void **ParseString** (dng\_stream &stream, dng\_string &s, CharSet charSet)
- void **SpoolString** (dng\_stream &stream, const dng\_string &s, uint8 dataSet, uint32 maxChars, CharSet charSet)
- bool **SafeForSystemEncoding** () const

### Static Protected Member Functions

- static bool **SafeForSystemEncoding** (const dng\_string &s)
- static bool **SafeForSystemEncoding** (const dng\_string\_list &list)

### 6.33.1 Detailed Description

Class for reading and holding IPTC metadata associated with a DNG file.

See the [IPTC specification](#) for information on member fields of this class.

### 6.33.2 Member Function Documentation

#### 6.33.2.1 `bool dng_iptc::IsEmpty () const`

Test if IPTC metadata exists.

**Return values:**

*true* if no IPTC metadata exists for this DNG.

References `NotEmpty()`.

Referenced by `NotEmpty()`.

#### 6.33.2.2 `bool dng_iptc::NotEmpty () const` `[inline]`

Test if IPTC metadata exists.

**Return values:**

*true* if IPTC metadata exists for this DNG.

References `IsEmpty()`.

Referenced by `IsEmpty()`.

#### 6.33.2.3 `void dng_iptc::Parse (const void * blockData, uint32 blockSize, uint64 offsetInOriginalFile)`

Parse a complete block of IPTC data.

**Parameters:**

*blockData* The block of IPTC data.

*blockSize* Size in bytes of data block.

*offsetInOriginalFile* Used to enable certain file patching operations such as updating date/time in place.

References `dng_stream::Get()`, `dng_stream::Get_int8()`, `dng_stream::Get_uint16()`, `dng_stream::Get_uint8()`, `dng_stream::Length()`, `dng_stream::Position()`, `dng_stream::SetBigEndian()`, and `dng_stream::SetReadPosition()`.

#### 6.33.2.4 dng\_memory\_block \* dng\_iptc::Spool (dng\_memory\_allocator & allocator, bool padForTIFF)

Serialize IPTC data to a memory block.

##### Parameters:

*allocator* Memory allocator used to acquire memory block.

*padForTIFF* Forces length of block to be a multiple of four bytes in accordance with TIFF standard.

##### Return values:

*Memory* block

References `dng_stream::AsMemoryBlock()`, `DNG_ASSERT`, `dng_stream::Flush()`, `dng_stream::Length()`, `dng_stream::Put()`, `dng_stream::Put_uint16()`, `dng_stream::Put_uint8()`, and `dng_stream::SetBigEndian()`.

The documentation for this class was generated from the following files:

- [dng\\_iptc.h](#)
- [dng\\_iptc.cpp](#)

## 6.34 dng\_linearization\_info Class Reference

Class for managing data values related to DNG linearization.

```
#include <dng_linearization_info.h>
```

### Public Member Functions

- void **RoundBlacks** ()
- virtual void **Parse** ([dng\\_host](#) &host, [dng\\_stream](#) &stream, [dng\\_info](#) &info)
- virtual void **PostParse** ([dng\\_host](#) &host, [dng\\_negative](#) &negative)
- real64 **MaxBlackLevel** (uint32 plane) const

*Compute the maximum black level for a given sample plane taking into account base black level, repeated black level pattenr, and row/column delta maps.*

- virtual void `Linearize` (`dng_host` &host, const `dng_image` &srcImage, `dng_image` &dstImage)
- `dng_urational BlackLevel` (uint32 row, uint32 col, uint32 plane) const
- uint32 `RowBlackCount` () const  
*Number of per-row black level deltas in `fBlackDeltaV`.*
- `dng_srational RowBlack` (uint32 row) const
- uint32 `ColumnBlackCount` () const  
*Number of per-column black level deltas in `fBlackDeltaV`.*
- `dng_srational ColumnBlack` (uint32 col) const

#### Public Attributes

- `dng_rect fActiveArea`
- uint32 `fMaskedAreaCount`  
*Number of rectangles in `fMaskedArea`.*
- `dng_rect fMaskedArea` [`kMaxMaskedAreas`]
- `AutoPtr`< `dng_memory_block` > `fLinearizationTable`
- uint32 `fBlackLevelRepeatRows`  
*Actual number of rows in `fBlackLevel` pattern.*
- uint32 `fBlackLevelRepeatCols`  
*Actual number of columns in `fBlackLevel` pattern.*
- real64 `fBlackLevel` [`kMaxBlackPattern`][`kMaxBlackPattern`][`kMaxSamplesPerPixel`]  
  
*Repeating pattern of black level deltas `fBlackLevelRepeatRows` by `fBlackLevelRepeatCols` in size.*
- `AutoPtr`< `dng_memory_block` > `fBlackDeltaH`  
*Memory block of double-precision floating point deltas between baseline black level and a given column's black level.*
- `AutoPtr`< `dng_memory_block` > `fBlackDeltaV`  
*Memory block of double-precision floating point deltas between baseline black level and a given row's black level.*
- real64 `fWhiteLevel` [`kMaxSamplesPerPixel`]  
*Single white level (maximum sensor value) for each sample plane.*

### Protected Attributes

- `int32 fBlackDenom`

#### 6.34.1 Detailed Description

Class for managing data values related to DNG linearization.

See `LinearizationTable`, `BlackLevel`, `BlackLevelRepeatDim`, `BlackLevelDeltaH`, `BlackLevelDeltaV` and `WhiteLevel` tags in the [DNG 1.1.0 specification](#).

#### 6.34.2 Member Function Documentation

##### 6.34.2.1 `dng_urational dng_linearization_info::BlackLevel (uint32 row, uint32 col, uint32 plane) const`

Compute black level for one coordinate and sample plane in the image.

#### Parameters:

- row* Row to compute black level for.
- col* Column to compute black level for.
- plane* Sample plane to compute black level for.

References `fBlackLevel`.

##### 6.34.2.2 `dng_srational dng_linearization_info::ColumnBlack (uint32 col) const`

Lookup black level delta for a given column.

#### Parameters:

- col* Column to get black level for.

#### Return values:

- black* level for indicated column.

References `fBlackDeltaH`, and `AutoPtr< T >::Get()`.

### 6.34.2.3 void dng\_linearization\_info::Linearize (dng\_host & *host*, const dng\_image & *srcImage*, dng\_image & *dstImage*) [virtual]

Convert raw data from in-file format to a true linear image using linearization data from DNG.

#### Parameters:

*host* Used to allocate buffers, check for aborts, and post progress updates.

*srcImage* Input pre-linearization RAW samples.

*dstImage* Output linearized image.

References fActiveArea, and dng\_host::PerformAreaTask().

### 6.34.2.4 dng\_srational dng\_linearization\_info::RowBlack (uint32 *row*) const

Lookup black level delta for a given row.

#### Parameters:

*row* Row to get black level for.

#### Return values:

*black* level for indicated row.

References fBlackDeltaV, and AutoPtr< T >::Get().

## 6.34.3 Member Data Documentation

### 6.34.3.1 dng\_rect dng\_linearization\_info::fActiveArea

This rectangle defines the active (non-masked) pixels of the sensor. The order of the rectangle coordinates is: top, left, bottom, right.

Referenced by Linearize().

### 6.34.3.2 AutoPtr<dng\_memory\_block> dng\_linearization\_info::fLinearizationTable

A lookup table that maps stored values into linear values. This tag is typically used to increase compression ratios by storing the raw data in a non-linear, more visually uniform space with fewer total encoding levels. If `SamplesPerPixel` is not equal to one, e.g. Fuji S3 type sensor, this single table applies to all the samples for each pixel.

Referenced by `dng_image_writer::WriteDNG()`.

### 6.34.3.3 `dng_rect dng_linearization_info::fMaskedArea[kMaxMaskedAreas]`

List of non-overlapping rectangle coordinates of fully masked pixels. Can be optionally used by DNG readers to measure the black encoding level. The order of each rectangle's coordinates is: top, left, bottom, right. If the raw image data has already had its black encoding level subtracted, then this tag should not be used, since the masked pixels are no longer useful. Note that DNG writers are still required to include an estimate and store the black encoding level using the black level DNG tags. Support for the `MaskedAreas` tag is not required of DNG readers.

The documentation for this class was generated from the following files:

- [dng\\_linearization\\_info.h](#)
- `dng_linearization_info.cpp`

## 6.35 `dng_memory_allocator` Class Reference

Interface for `dng_memory_block` allocator.

```
#include <dng_memory.h>
```

### Public Member Functions

- virtual `dng_memory_block * Allocate (uint32 size)`

#### 6.35.1 Detailed Description

Interface for `dng_memory_block` allocator.

#### 6.35.2 Member Function Documentation

- ##### 6.35.2.1 `dng_memory_block * dng_memory_allocator::Allocate (uint32 size)` [virtual]

Allocate a `dng_memory` block.

**Parameters:**

*size* Number of bytes in memory block.

**Return values:**

A `dng_memory_block` with at least *size* bytes of valid storage.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_memory`.

References `ThrowMemoryFull()`.

Referenced by `dng_host::Allocate()`, `dng_stream::AsMemoryBlock()`, `dng_1d_table::Initialize()`, and `dng_filter_task::Start()`.

The documentation for this class was generated from the following files:

- `dng_memory.h`
- `dng_memory.cpp`

## 6.36 `dng_memory_block` Class Reference

Class to provide resource acquisition is instantiation discipline for image buffers and other larger memory allocations.

```
#include <dng_memory.h>
```

Inherited by `dng_malloc_block`.

**Public Member Functions**

- `uint32 LogicalSize () const`
- `void * Buffer ()`
- `const void * Buffer () const`
- `char * Buffer_char ()`
- `const char * Buffer_char () const`
- `uint8 * Buffer_uint8 ()`
- `const uint8 * Buffer_uint8 () const`
- `uint16 * Buffer_uint16 ()`
- `const uint16 * Buffer_uint16 () const`
- `int16 * Buffer_int16 ()`
- `const int16 * Buffer_int16 () const`

- `uint32 * Buffer_uint32 ()`
- `const uint32 * Buffer_uint32 () const`
- `int32 * Buffer_int32 ()`
- `const int32 * Buffer_int32 () const`
- `real32 * Buffer_real32 ()`
- `const real32 * Buffer_real32 () const`
- `real64 * Buffer_real64 ()`
- `const real64 * Buffer_real64 () const`

### Protected Member Functions

- `dng_memory_block (uint32 logicalSize)`
- `uint32 PhysicalSize ()`
- `void SetBuffer (void *p)`

#### 6.36.1 Detailed Description

Class to provide resource acquisition is instantiation discipline for image buffers and other larger memory allocations.

This class requires a [dng\\_memory\\_allocator](#) for allocation.

#### 6.36.2 Member Function Documentation

##### 6.36.2.1 `const void* dng_memory_block::Buffer () const` [inline]

Return pointer to allocated memory as a `const void *`.

#### Return values:

*const void \** valid for as many bytes as were allocated.

##### 6.36.2.2 `void* dng_memory_block::Buffer ()` [inline]

Return pointer to allocated memory as a `void *`.

#### Return values:

*void \** valid for as many bytes as were allocated.

Referenced by `Buffer_char()`, `Buffer_int16()`, `Buffer_int32()`, `Buffer_real32()`, `Buffer_real64()`, `Buffer_uint16()`, `Buffer_uint32()`, and `Buffer_uint8()`.

**6.36.2.3** `const char* dng_memory_block::Buffer_char () const` [inline]

Return pointer to allocated memory as a `const char *`.

**Return values:**

*const* `char *` valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.4** `char* dng_memory_block::Buffer_char ()` [inline]

Return pointer to allocated memory as a `char *`.

**Return values:**

*char \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.5** `const int16* dng_memory_block::Buffer_int16 () const` [inline]

Return pointer to allocated memory as a `const int16 *`.

**Return values:**

*const* `int16 *` valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.6** `int16* dng_memory_block::Buffer_int16 ()` [inline]

Return pointer to allocated memory as a `int16 *`.

**Return values:**

*int16 \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.7** `const int32* dng_memory_block::Buffer_int32 () const` [inline]

Return pointer to allocated memory as a `const int32 *`.

**Return values:**

*const int32 \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.8** `int32* dng_memory_block::Buffer_int32 ()` [inline]

Return pointer to allocated memory as a `int32 *`.

**Return values:**

*int32 \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.9** `const real32* dng_memory_block::Buffer_real32 () const` [inline]

Return pointer to allocated memory as a `const real32 *`.

**Return values:**

*const real32 \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.10** `real32* dng_memory_block::Buffer_real32 ()` [inline]

Return pointer to allocated memory as a `real32 *`.

**Return values:**

*real32 \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.11** `const real64* dng_memory_block::Buffer_real64 () const`  
[inline]

Return pointer to allocated memory as a `const real64 *`.

**Return values:**

*const* `real64 *` valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.12** `real64* dng_memory_block::Buffer_real64 ()` [inline]

Return pointer to allocated memory as a `real64 *`.

**Return values:**

*real64 \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.13** `const uint16* dng_memory_block::Buffer_uint16 () const`  
[inline]

Return pointer to allocated memory as a `const uint16 *`.

**Return values:**

*const* `uint16 *` valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.14** `uint16* dng_memory_block::Buffer_uint16 ()` [inline]

Return pointer to allocated memory as a `uint16 *`.

**Return values:**

*uint16 \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.15** `const uint32* dng_memory_block::Buffer_uint32 () const` `[inline]`

Return pointer to allocated memory as a `const uint32 *`.

**Return values:**

*const* `uint32 *` valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.16** `uint32* dng_memory_block::Buffer_uint32 ()` `[inline]`

Return pointer to allocated memory as a `uint32 *`.

**Return values:**

*uint32 \** valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.17** `const uint8* dng_memory_block::Buffer_uint8 () const` `[inline]`

Return pointer to allocated memory as a `const uint8 *`.

**Return values:**

*const* `uint8 *` valid for as many bytes as were allocated.

References `Buffer()`.

**6.36.2.18** `uint8* dng_memory_block::Buffer_uint8 ()` `[inline]`

Return pointer to allocated memory as a `uint8 *`.

**Return values:**

*uint8 \** valid for as many bytes as were allocated.

References `Buffer()`.

Referenced by `dng_memory_stream::CopyToStream()`.

### 6.36.2.19 uint32 dng\_memory\_block::LogicalSize () const [inline]

Getter for available size, in bytes, of memory block.

#### Return values:

*size* in bytes of available memory in memory block.

The documentation for this class was generated from the following file:

- dng\_memory.h

## 6.37 dng\_memory\_data Class Reference

Class to provide resource acquisition is instantiation discipline for small memory allocations.

```
#include <dng_memory.h>
```

#### Public Member Functions

- [dng\\_memory\\_data \(\)](#)
- [dng\\_memory\\_data \(uint32 size\)](#)
- [~dng\\_memory\\_data \(\)](#)  
*Release memory buffer using free.*
- void [Allocate](#) (uint32 size)
- void [Clear](#) ()
- void \* [Buffer](#) ()
- const void \* [Buffer](#) () const
- char \* [Buffer\\_char](#) ()
- const char \* [Buffer\\_char](#) () const
- uint8 \* [Buffer\\_uint8](#) ()
- const uint8 \* [Buffer\\_uint8](#) () const
- uint16 \* [Buffer\\_uint16](#) ()
- const uint16 \* [Buffer\\_uint16](#) () const
- int16 \* [Buffer\\_int16](#) ()
- const int16 \* [Buffer\\_int16](#) () const
- uint32 \* [Buffer\\_uint32](#) ()
- const uint32 \* [Buffer\\_uint32](#) () const
- int32 \* [Buffer\\_int32](#) ()
- const int32 \* [Buffer\\_int32](#) () const

- `uint64 * Buffer_uint64 ()`
- `const uint64 * Buffer_uint64 () const`
- `int64 * Buffer_int64 ()`
- `const int64 * Buffer_int64 () const`
- `real32 * Buffer_real32 ()`
- `const real32 * Buffer_real32 () const`
- `real64 * Buffer_real64 ()`
- `const real64 * Buffer_real64 () const`

### 6.37.1 Detailed Description

Class to provide resource acquisition is instantiation discipline for small memory allocations.

Support for memory allocation. This class does not use `dng_memory_allocator` for memory allocation.

### 6.37.2 Constructor & Destructor Documentation

#### 6.37.2.1 `dng_memory_data::dng_memory_data ()`

Construct an empty memory buffer using malloc.

#### Exceptions:

*dng\_memory\_full* with `fErrorCode` equal to `dng_error_memory`.

#### 6.37.2.2 `dng_memory_data::dng_memory_data (uint32 size)`

Construct memory buffer of size bytes using malloc.

#### Parameters:

*size* Number of bytes of memory needed.

#### Exceptions:

*dng\_memory\_full* with `fErrorCode` equal to `dng_error_memory`.

References `Allocate()`.

### 6.37.3 Member Function Documentation

#### 6.37.3.1 `void dng_memory_data::Allocate (uint32 size)`

Clear existing memory buffer and allocate new memory of size bytes.

**Parameters:**

*size* Number of bytes of memory needed.

**Exceptions:**

*dng\_memory\_full* with `fErrorCode` equal to `dng_error_memory`.

References `Clear()`, and `ThrowMemoryFull()`.

Referenced by `dng_memory_data()`.

#### 6.37.3.2 `const void* dng_memory_data::Buffer () const` [inline]

Return pointer to allocated memory as a `const void *`.

**Return values:**

*const void \** valid for as many bytes as were allocated.

#### 6.37.3.3 `void* dng_memory_data::Buffer ()` [inline]

Return pointer to allocated memory as a `void *`.

**Return values:**

*void \** valid for as many bytes as were allocated.

Referenced by `Buffer_char()`, `Buffer_int16()`, `Buffer_int32()`, `Buffer_int64()`, `Buffer_real32()`, `Buffer_real64()`, `Buffer_uint16()`, `Buffer_uint32()`, `Buffer_uint64()`, `Buffer_uint8()`, `dng_stream::CopyToStream()`, and `dng_stream::PutZeros()`.

#### 6.37.3.4 `const char* dng_memory_data::Buffer_char () const` [inline]

Return pointer to allocated memory as a `const char *`.

**Return values:**

*const* char \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.5 char\* dng\_memory\_data::Buffer\_char () [inline]**

Return pointer to allocated memory as a char \*.

**Return values:**

*char* \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.6 const int16\* dng\_memory\_data::Buffer\_int16 () const [inline]**

Return pointer to allocated memory as a const int16 \*.

**Return values:**

*const* int16 \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.7 int16\* dng\_memory\_data::Buffer\_int16 () [inline]**

Return pointer to allocated memory as a int16 \*.

**Return values:**

*int16* \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.8 const int32\* dng\_memory\_data::Buffer\_int32 () const [inline]**

Return pointer to allocated memory as a const int32 \*.

**Return values:**

*const* `int32*` valid for as many bytes as were allocated.

References `Buffer()`.

**6.37.3.9** `int32* dng_memory_data::Buffer_int32 ()` [inline]

Return pointer to allocated memory as a `const int32*`.

**Return values:**

*const* `int32*` valid for as many bytes as were allocated.

References `Buffer()`.

**6.37.3.10** `const int64* dng_memory_data::Buffer_int64 () const` [inline]

Return pointer to allocated memory as a `const int64*`.

**Return values:**

*const* `int64*` valid for as many bytes as were allocated.

References `Buffer()`.

**6.37.3.11** `int64* dng_memory_data::Buffer_int64 ()` [inline]

Return pointer to allocated memory as a `const int64*`.

**Return values:**

*const* `int64*` valid for as many bytes as were allocated.

References `Buffer()`.

**6.37.3.12** `const real32* dng_memory_data::Buffer_real32 () const` [inline]

Return pointer to allocated memory as a `const real32*`.

**Return values:**

*const* `real32` \* valid for as many bytes as were allocated.

References `Buffer()`.

**6.37.3.13** `real32* dng_memory_data::Buffer_real32 ()` `[inline]`

Return pointer to allocated memory as a `real32` \*.

**Return values:**

*real32* \* valid for as many bytes as were allocated.

References `Buffer()`.

**6.37.3.14** `const real64* dng_memory_data::Buffer_real64 () const` `[inline]`

Return pointer to allocated memory as a `const real64` \*.

**Return values:**

*const* `real64` \* valid for as many bytes as were allocated.

References `Buffer()`.

**6.37.3.15** `real64* dng_memory_data::Buffer_real64 ()` `[inline]`

Return pointer to allocated memory as a `real64` \*.

**Return values:**

*real64* \* valid for as many bytes as were allocated.

References `Buffer()`.

**6.37.3.16** `const uint16* dng_memory_data::Buffer_uint16 () const` `[inline]`

Return pointer to allocated memory as a `const uint16` \*.

**Return values:**

*const* uint16 \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.17 uint16\* dng\_memory\_data::Buffer\_uint16 () [inline]**

Return pointer to allocated memory as a uint16 \*.

**Return values:**

*uint16* \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.18 const uint32\* dng\_memory\_data::Buffer\_uint32 () const [inline]**

Return pointer to allocated memory as a uint32 \*.

**Return values:**

*uint32* \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.19 uint32\* dng\_memory\_data::Buffer\_uint32 () [inline]**

Return pointer to allocated memory as a uint32 \*.

**Return values:**

*uint32* \* valid for as many bytes as were allocated.

References Buffer().

Referenced by dng\_image\_writer::WriteDNG().

**6.37.3.20 const uint64\* dng\_memory\_data::Buffer\_uint64 () const [inline]**

Return pointer to allocated memory as a uint64 \*.

**Return values:**

*uint64* \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.21 uint64\* dng\_memory\_data::Buffer\_uint64 () [inline]**

Return pointer to allocated memory as a uint64 \*.

**Return values:**

*uint64* \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.22 const uint8\* dng\_memory\_data::Buffer\_uint8 () const [inline]**

Return pointer to allocated memory as a const uint8 \*.

**Return values:**

*const* uint8 \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.23 uint8\* dng\_memory\_data::Buffer\_uint8 () [inline]**

Return pointer to allocated memory as a uint8 \*.

**Return values:**

*uint8* \* valid for as many bytes as were allocated.

References Buffer().

**6.37.3.24 void dng\_memory\_data::Clear ()**

Release any allocated memory using free. Object is still valid and Allocate can be called again.

Referenced by `Allocate()`, and `~dng_memory_data()`.

The documentation for this class was generated from the following files:

- `dng_memory.h`
- `dng_memory.cpp`

## 6.38 `dng_memory_stream` Class Reference

A `dng_stream` which can be read from or written to memory.

```
#include <dng_memory_stream.h>
```

Inheritance diagram for `dng_memory_stream`:

### Public Member Functions

- `dng_memory_stream` (`dng_memory_allocator` &allocator, `dng_abort_sniffer` \*sniffer=NULL, uint32 pageSize=64 \*1024)
- virtual void `CopyToStream` (`dng_stream` &dstStream, uint64 count)

### Protected Member Functions

- virtual uint64 `DoGetLength` ()
- virtual void `DoRead` (void \*data, uint32 count, uint64 offset)
- virtual void `DoSetLength` (uint64 length)
- virtual void `DoWrite` (const void \*data, uint32 count, uint64 offset)

### Protected Attributes

- `dng_memory_allocator` & `fAllocator`
- uint32 `fPageSize`
- uint32 `fPageCount`
- uint32 `fPagesAllocated`
- `dng_memory_block` \*\* `fPageList`
- uint64 `fMemoryStreamLength`

#### 6.38.1 Detailed Description

A `dng_stream` which can be read from or written to memory.

Stream is populated via writing and either read or accessed by asking for contents as a pointer.

## 6.38.2 Constructor & Destructor Documentation

### 6.38.2.1 dng\_memory\_stream::dng\_memory\_stream (dng\_memory\_allocator & allocator, dng\_abort\_sniffer \* sniffer = NULL, uint32 pageSize = 64 \* 1024)

Construct a new memory-based stream.

#### Parameters:

*allocator* Allocator to use to allocate memory in stream as needed.

*sniffer* If non-NULL used to check for user cancellation.

*pageSize* Unit of allocation for data stored in stream.

## 6.38.3 Member Function Documentation

### 6.38.3.1 void dng\_memory\_stream::CopyToStream (dng\_stream & dstStream, uint64 count) [virtual]

Copy a specified number of bytes to a target stream.

#### Parameters:

*dstStream* The target stream.

*count* The number of bytes to copy.

Reimplemented from [dng\\_stream](#).

References [dng\\_memory\\_block::Buffer\\_uint8\(\)](#), [dng\\_stream::Flush\(\)](#), [dng\\_stream::Length\(\)](#), [dng\\_stream::Position\(\)](#), [dng\\_stream::Put\(\)](#), [dng\\_stream::SetReadPosition\(\)](#), and [ThrowEndOfFile\(\)](#).

The documentation for this class was generated from the following files:

- [dng\\_memory\\_stream.h](#)
- [dng\\_memory\\_stream.cpp](#)

## 6.39 dng\_mosaic\_info Class Reference

Support for describing color filter array patterns and manipulating mosaic sample data.

```
#include <dng_mosaic_info.h>
```

**Public Member Functions**

- virtual void **Parse** (dng\_host &host, dng\_stream &stream, dng\_info &info)
- virtual void **PostParse** (dng\_host &host, dng\_negative &negative)
- bool **IsColorFilterArray** () const
- virtual bool **SetFourColorBayer** ()
- virtual dng\_point **FullScale** () const
- virtual dng\_point **DownScale** (uint32 minSize, uint32 prefSize, real64 cropFactor) const
- virtual dng\_point **DstSize** (const dng\_point &downScale) const
- virtual void **InterpolateGeneric** (dng\_host &host, dng\_negative &negative, const dng\_image &srcImage, dng\_image &dstImage, uint32 srcPlane=0) const
- virtual void **InterpolateFast** (dng\_host &host, dng\_negative &negative, const dng\_image &srcImage, dng\_image &dstImage, const dng\_point &downScale, uint32 srcPlane=0) const
- virtual void **Interpolate** (dng\_host &host, dng\_negative &negative, const dng\_image &srcImage, dng\_image &dstImage, const dng\_point &downScale, uint32 srcPlane=0) const

**Public Attributes**

- dng\_point **fCFAPatternSize**  
*Size of fCFAPattern.*
- uint8 **fCFAPattern** [kMaxCFAPattern][kMaxCFAPattern]  
*CFA pattern from CFAPattern tag in the TIFF/EP specification..*
- uint32 **fColorPlanes**  
*Number of color planes in DNG input.*
- uint8 **fCFAPlaneColor** [kMaxColorPlanes]
- uint32 **fCFALayout**
- uint32 **fBayerGreenSplit**

**Protected Member Functions**

- virtual bool **IsSafeDownScale** (const dng\_point &downScale) const
- uint32 **SizeForDownScale** (const dng\_point &downScale) const
- virtual bool **ValidSizeDownScale** (const dng\_point &downScale, uint32 minSize) const

### Protected Attributes

- dng\_point **fSrcSize**
- dng\_point **fCroppedSize**
- real64 **fAspectRatio**

### 6.39.1 Detailed Description

Support for describing color filter array patterns and manipulating mosaic sample data.

See CFAPattern tag in [TIFF/EP specification](#) and CFAPlaneColor, CFALayout, and BayerGreenSplit tags in the [DNG 1.1.0 specification](#).

### 6.39.2 Member Function Documentation

#### 6.39.2.1 dng\_point dng\_mosaic\_info::DownScale (uint32 *minSize*, uint32 *prefSize*, real64 *cropFactor*) const [virtual]

Returns integer factors by which mosaic data must be downsampled to produce an image which is as close to *prefSize* as possible in longer dimension, but no smaller than *minSize*.

#### Parameters:

*minSize* Number of pixels as minimum for longer dimension of downsampled image.

*prefSize* Number of pixels as target for longer dimension of downsampled image.

*cropFactor* Fraction of the image to be used after cropping.

#### Return values:

*Point* containing integer factors by which image must be downsampled.

References IsColorFilterArray().

#### 6.39.2.2 dng\_point dng\_mosaic\_info::DstSize (const dng\_point & *downScale*) const [virtual]

Return size of demosaiced image for passed in downscaling factor.

#### Parameters:

*downScale* Integer downsampling factor obtained from DownScale method.

**Return values:**

*Size* of resulting demosaiced image.

References FullScale().

**6.39.2.3 dng\_point dng\_mosaic\_info::FullScale () const [virtual]**

Returns scaling factor relative to input size needed to capture output data. Staggered (or rotated) sensing arrays are produced to a larger output than the number of input samples. This method indicates how much larger.

**Return values:**

*a* point with integer scaling factors for the horizontal and vertical dimensions.

References fCFALayout.

Referenced by DstSize(), and InterpolateGeneric().

**6.39.2.4 void dng\_mosaic\_info::Interpolate (dng\_host & host, dng\_negative & negative, const dng\_image & srcImage, dng\_image & dstImage, const dng\_point & downScale, uint32 srcPlane = 0) const [virtual]**

Demosaic interpolation of a single plane. Chooses between generic and fast interpolators based on parameters.

**Parameters:**

*host* [dng\\_host](#) to use for buffer allocation requests, user cancellation testing, and progress updates.

*negative* DNG negative of mosaiced data.

*srcImage* Source image for mosaiced data.

*dstImage* Destination image for resulting interpolated data.

*downScale* Amount (in horizontal and vertical) by which to subsample image.

*srcPlane* Which plane to interpolate.

References InterpolateFast(), and InterpolateGeneric().

**6.39.2.5** `void dng_mosaic_info::InterpolateFast (dng_host & host, dng_negative & negative, const dng_image & srcImage, dng_image & dstImage, const dng_point & downScale, uint32 srcPlane = 0) const` [virtual]

Demosaic interpolation of a single plane for downsampled case.

**Parameters:**

*host* `dng_host` to use for buffer allocation requests, user cancellation testing, and progress updates.

*negative* DNG negative of mosaiced data.

*srcImage* Source image for mosaiced data.

*dstImage* Destination image for resulting interpolated data.

*downScale* Amount (in horizontal and vertical) by which to subsample image.

*srcPlane* Which plane to interpolate.

References `dng_image::Bounds()`, and `dng_host::PerformAreaTask()`.

Referenced by `Interpolate()`.

**6.39.2.6** `void dng_mosaic_info::InterpolateGeneric (dng_host & host, dng_negative & negative, const dng_image & srcImage, dng_image & dstImage, uint32 srcPlane = 0) const` [virtual]

Demosaic interpolation of a single plane for non-downsampled case.

**Parameters:**

*host* `dng_host` to use for buffer allocation requests, user cancellation testing, and progress updates.

*negative* DNG negative of mosaiced data.

*srcImage* Source image for mosaiced data.

*dstImage* Destination image for resulting interpolated data.

*srcPlane* Which plane to interpolate.

References `dng_host::Allocate()`, `dng_image::Bounds()`, `dng_image::edge_repeat`, `fCFAPatternSize`, `FullScale()`, `dng_image::Get()`, `dng_image::PixelSize()`, `dng_image::PixelType()`, `dng_image::Put()`, `dng_image::RepeatingTile()`, and `dng_host::SniffForAbort()`.

Referenced by `Interpolate()`.

### 6.39.2.7 bool dng\_mosaic\_info::IsColorFilterArray () const [inline]

Returns whether the RAW data in this DNG file from a color filter array (mosaic) source.

#### Return values:

*true* if this DNG file is from a color filter array (mosaic) source.

References fCFAPatternSize.

Referenced by DownScale(), and dng\_image\_writer::WriteDNG().

### 6.39.2.8 bool dng\_mosaic\_info::SetFourColorBayer () [virtual]

Enable generating four-plane output from three-plane Bayer input. Extra plane is a second version of the green channel. First green is produced using green mosaic samples from one set of rows/columns (even/odd) and the second green channel is produced using the other set of rows/columns. One can compare the two versions to judge whether BayerGreenSplit needs to be set for a given input source.

References fCFAPattern, fCFAPatternSize, and fColorPlanes.

## 6.39.3 Member Data Documentation

### 6.39.3.1 uint32 dng\_mosaic\_info::fBayerGreenSplit

Value of BayerGreenSplit tag in DNG file. BayerGreenSplit only applies to CFA images using a Bayer pattern filter array. This tag specifies, in arbitrary units, how closely the values of the green pixels in the blue/green rows track the values of the green pixels in the red/green rows.

A value of zero means the two kinds of green pixels track closely, while a non-zero value means they sometimes diverge. The useful range for this tag is from 0 (no divergence) to about 5000 (large divergence).

### 6.39.3.2 uint32 dng\_mosaic\_info::fCFALayout

Value of CFALayout tag in the [DNG 1.3 specification](#). CFALayout describes the spatial layout of the CFA. The currently defined values are:

- 1 = Rectangular (or square) layout.
- 2 = Staggered layout A: even columns are offset down by 1/2 row.
- 3 = Staggered layout B: even columns are offset up by 1/2 row.
- 4 = Staggered layout C: even rows are offset right by 1/2 column.
- 5 = Staggered layout D: even rows are offset left by 1/2 column.
- 6 = Staggered layout E: even rows are offset up by 1/2 row, even columns are offset left by 1/2 column.
- 7 = Staggered layout F: even rows are offset up by 1/2 row, even columns are offset right by 1/2 column.
- 8 = Staggered layout G: even rows are offset down by 1/2 row, even columns are offset left by 1/2 column.
- 9 = Staggered layout H: even rows are offset down by 1/2 row, even columns are offset right by 1/2 column.

Referenced by `FullScale()`, and `dng_image_writer::WriteDNG()`.

The documentation for this class was generated from the following files:

- [dng\\_mosaic\\_info.h](#)
- `dng_mosaic_info.cpp`

## 6.40 `dng_negative` Class Reference

Main class for holding DNG image data and associated metadata.

```
#include <dng_negative.h>
```

### Public Types

- enum `RawImageStageEnum` {  
    `rawImageStagePreOpcode1`,   `rawImageStagePostOpcode1`,   `rawImageStagePostOpcode2`,  
    `rawImageStagePreOpcode3`,  
    `rawImageStagePostOpcode3`, `rawImageStageNone` }

### Public Member Functions

- [dng\\_memory\\_allocator](#) & [Allocator](#) () const  
*Provide access to the memory allocator used for this object.*
- void [SetModelName](#) (const char \*name)  
*Getter for ModelName.*
- const dng\_string & [ModelName](#) () const  
*Setter for ModelName.*
- void [SetLocalName](#) (const char \*name)  
*Setter for LocalName.*
- const dng\_string & [LocalName](#) () const  
*Getter for LocalName.*
- void [SetBaseOrientation](#) (const dng\_orientation &orientation)  
*Setter for BaseOrientation.*
- bool [HasBaseOrientation](#) () const  
*Has BaseOrientation been set?*
- const dng\_orientation & [BaseOrientation](#) () const  
*Getter for BaseOrientation.*
- virtual dng\_orientation [Orientation](#) () const  
*Hook to allow SDK host code to add additional rotations.*
- void [ApplyOrientation](#) (const dng\_orientation &orientation)
- void [SetDefaultCropSize](#) (const dng\_urational &sizeH, const dng\_urational &sizeV)  
*Setter for DefaultCropSize.*
- void [SetDefaultCropSize](#) (uint32 sizeH, uint32 sizeV)  
*Setter for DefaultCropSize.*
- const dng\_urational & [DefaultCropSizeH](#) () const  
*Getter for DefaultCropSize horizontal.*
- const dng\_urational & [DefaultCropSizeV](#) () const  
*Getter for DefaultCropSize vertical.*

- void [SetDefaultCropOrigin](#) (const dng\_urational &originH, const dng\_urational &originV)  
*Setter for DefaultCropOrigin.*
- void [SetDefaultCropOrigin](#) (uint32 originH, uint32 originV)  
*Setter for DefaultCropOrigin.*
- void [SetDefaultCropCentered](#) (const dng\_point &rawSize)  
*Set default crop around center of image.*
- const dng\_urational & [DefaultCropOriginH](#) () const  
*Get default crop origin horizontal value.*
- const dng\_urational & [DefaultCropOriginV](#) () const  
*Get default crop origin vertical value.*
- void [SetDefaultScale](#) (const dng\_urational &scaleH, const dng\_urational &scaleV)  
*Setter for DefaultScale.*
- const dng\_urational & [DefaultScaleH](#) () const  
*Get default scale horizontal value.*
- const dng\_urational & [DefaultScaleV](#) () const  
*Get default scale vertical value.*
- void [SetBestQualityScale](#) (const dng\_urational &scale)  
*Setter for BestQualityScale.*
- const dng\_urational & [BestQualityScale](#) () const  
*Getter for BestQualityScale.*
- real64 [RawToFullScaleH](#) () const  
*API for raw to full image scaling factors horizontal.*
- real64 [RawToFullScaleV](#) () const  
*API for raw to full image scaling factors vertical.*
- real64 [DefaultScale](#) () const
- real64 [SquareWidth](#) () const  
*Default cropped image size (at scale == 1.0) width.*
- real64 [SquareHeight](#) () const

*Default cropped image size (at scale == 1.0) height.*

- real64 [AspectRatio](#) () const  
*Default cropped image aspect ratio.*
- real64 [PixelAspectRatio](#) () const  
*Pixel aspect ratio of stage 3 image.*
- uint32 [FinalWidth](#) (real64 scale) const  
*Default cropped image size at given scale factor width.*
- uint32 [FinalHeight](#) (real64 scale) const  
*Default cropped image size at given scale factor height.*
- uint32 [DefaultFinalWidth](#) () const  
*Default cropped image size at default scale factor width.*
- uint32 [DefaultFinalHeight](#) () const  
*Default cropped image size at default scale factor height.*
- uint32 [BestQualityFinalWidth](#) () const
- uint32 [BestQualityFinalHeight](#) () const
- dng\_rect [DefaultCropArea](#) (real64 scaleH=1.0, real64 scaleV=1.0) const
- void [SetBaselineNoise](#) (real64 noise)  
*Setter for BaselineNoise.*
- const dng\_urational & [BaselineNoiseR](#) () const  
*Getter for BaselineNoise as dng\_urational.*
- real64 [BaselineNoise](#) () const  
*Getter for BaselineNoise as real64.*
- void [SetNoiseReductionApplied](#) (const dng\_urational &value)  
*Setter for NoiseReductionApplied.*
- const dng\_urational & [NoiseReductionApplied](#) () const  
*Getter for NoiseReductionApplied.*
- void [SetNoiseProfile](#) (const [dng\\_noise\\_profile](#) &noiseProfile)  
*Setter for noise profile.*
- bool [HasNoiseProfile](#) () const

*Does this negative have a valid noise profile?*

- const [dng\\_noise\\_profile](#) & [NoiseProfile](#) () const  
*Getter for noise profile.*
- void [SetBaselineExposure](#) (real64 exposure)  
*Setter for BaselineExposure.*
- const dng\_urational & [BaselineExposureR](#) () const  
*Getter for BaselineExposure as dng\_urational.*
- real64 [BaselineExposure](#) () const  
*Getter for BaselineExposure as real64.*
- void [SetBaselineSharpness](#) (real64 sharpness)  
*Setter for BaselineSharpness.*
- const dng\_urational & [BaselineSharpnessR](#) () const  
*Getter for BaselineSharpness as dng\_urational.*
- real64 [BaselineSharpness](#) () const  
*Getter for BaselineSharpness as real64.*
- void [SetChromaBlurRadius](#) (const dng\_urational &radius)  
*Setter for ChromaBlurRadius.*
- const dng\_urational & [ChromaBlurRadius](#) () const  
*Getter for ChromaBlurRadius as dng\_urational.*
- void [SetAntiAliasStrength](#) (const dng\_urational &strength)  
*Setter for AntiAliasStrength.*
- const dng\_urational & [AntiAliasStrength](#) () const  
*Getter for AntiAliasStrength as dng\_urational.*
- void [SetLinearResponseLimit](#) (real64 limit)  
*Setter for LinearResponseLimit.*
- const dng\_urational & [LinearResponseLimitR](#) () const  
*Getter for LinearResponseLimit as dng\_urational.*
- real64 [LinearResponseLimit](#) () const

*Getter for LinearResponseLimit as real64.*

- void **SetShadowScale** (const dng\_urational &scale)  
*Setter for ShadowScale.*
- const dng\_urational & **ShadowScaleR** () const  
*Getter for ShadowScale as dng\_urational.*
- real64 **ShadowScale** () const  
*Getter for ShadowScale as real64.*
- void **SetColorimetricReference** (uint32 ref)
- uint32 **ColorimetricReference** () const
- void **SetColorChannels** (uint32 channels)  
*Setter for ColorChannels.*
- uint32 **ColorChannels** () const  
*Getter for ColorChannels.*
- void **SetMonochrome** ()  
*Setter for Monochrome.*
- bool **IsMonochrome** () const  
*Getter for Monochrome.*
- void **SetAnalogBalance** (const dng\_vector &b)  
*Setter for AnalogBalance.*
- dng\_urational **AnalogBalanceR** (uint32 channel) const  
*Getter for AnalogBalance as dng\_urational.*
- real64 **AnalogBalance** (uint32 channel) const  
*Getter for AnalogBalance as real64.*
- void **SetCameraNeutral** (const dng\_vector &n)  
*Setter for CameraNeutral.*
- void **ClearCameraNeutral** ()  
*Clear CameraNeutral.*
- bool **HasCameraNeutral** () const  
*Determine if CameraNeutral has been set but not cleared.*

- const dng\_vector & [CameraNeutral](#) () const  
*Getter for CameraNeutral.*
- dng\_urational **CameraNeutralR** (uint32 channel) const
- void [SetCameraWhiteXY](#) (const dng\_xy\_coord &coord)  
*Setter for CameraWhiteXY.*
- bool **HasCameraWhiteXY** () const
- const dng\_xy\_coord & **CameraWhiteXY** () const
- void [GetCameraWhiteXY](#) (dng\_urational &x, dng\_urational &y) const
- void [SetCameraCalibration1](#) (const dng\_matrix &m)
- void [SetCameraCalibration2](#) (const dng\_matrix &m)
- const dng\_matrix & [CameraCalibration1](#) () const  
*Getter for first of up to two color matrices used for individual camera calibrations.*
- const dng\_matrix & [CameraCalibration2](#) () const  
*Getter for second of up to two color matrices used for individual camera calibrations.*
- void [SetCameraCalibrationSignature](#) (const char \*signature)
- const dng\_string & **CameraCalibrationSignature** () const
- void [AddProfile](#) ([AutoPtr](#)< [dng\\_camera\\_profile](#) > &profile)
- void **ClearProfiles** ()
- uint32 **ProfileCount** () const
- const [dng\\_camera\\_profile](#) & **ProfileByIndex** (uint32 index) const
- const [dng\\_camera\\_profile](#) \* **ProfileByID** (const dng\_camera\_profile\_id &id, bool useDefaultIfNoMatch=true) const
- bool **HasProfileID** (const dng\_camera\_profile\_id &id) const
- virtual const [dng\\_camera\\_profile](#) \* **CameraProfileToEmbed** () const
- void [SetAsShotProfileName](#) (const char \*name)
- const dng\_string & **AsShotProfileName** () const
- virtual [dng\\_color\\_spec](#) \* **MakeColorSpec** (const dng\_camera\_profile\_id &id) const
- void [SetRawImageDigest](#) (const [dng\\_fingerprint](#) &digest)
- void **ClearRawImageDigest** ()
- const [dng\\_fingerprint](#) & **RawImageDigest** () const
- void [FindRawImageDigest](#) ([dng\\_host](#) &host) const
- void [ValidateRawImageDigest](#) ([dng\\_host](#) &host)
- void [SetRawDataUniqueID](#) (const [dng\\_fingerprint](#) &id)
- const [dng\\_fingerprint](#) & **RawDataUniqueID** () const
- void [FindRawDataUniqueID](#) ([dng\\_host](#) &host) const
- void [RecomputeRawDataUniqueID](#) ([dng\\_host](#) &host)
- void [SetOriginalRawFileName](#) (const char \*name)
- bool **HasOriginalRawFileName** () const

- const dng\_string & **OriginalRawFileName** () const
- void **SetHasOriginalRawFileData** (bool hasData)
- bool **CanEmbedOriginalRaw** () const
- void **SetOriginalRawFileData** (AutoPtr< dng\_memory\_block > &data)
- const void \* **OriginalRawFileData** () const
- uint32 **OriginalRawFileDataLength** () const
- void **SetOriginalRawFileDigest** (const dng\_fingerprint &digest)
- const dng\_fingerprint & **OriginalRawFileDigest** () const
- void **FindOriginalRawFileDigest** () const
- void **ValidateOriginalRawFileDigest** ()
- void **SetPrivateData** (AutoPtr< dng\_memory\_block > &block)
- void **ClearPrivateData** ()
- const uint8 \* **PrivateData** () const
- uint32 **PrivateLength** () const
- void **SetMakerNoteSafety** (bool safe)
- bool **IsMakerNoteSafe** () const
- void **SetMakerNote** (AutoPtr< dng\_memory\_block > &block)
- void **ClearMakerNote** ()
- const void \* **MakerNoteData** () const
- uint32 **MakerNoteLength** () const
- dng\_exif \* **GetExif** ()
- const dng\_exif \* **GetExif** () const
- virtual dng\_memory\_block \* **BuildExifBlock** (const dng\_resolution \*resolution=NULL, bool includeIPTC=false, bool minimalEXIF=false, const dng\_jpeg\_preview \*thumbnail=NULL) const
- dng\_exif \* **GetOriginalExif** ()
- const dng\_exif \* **GetOriginalExif** () const
- void **SetIPTC** (AutoPtr< dng\_memory\_block > &block, uint64 offset)
- void **SetIPTC** (AutoPtr< dng\_memory\_block > &block)
- void **ClearIPTC** ()
- const void \* **IPTCData** () const
- uint32 **IPTCLength** () const
- uint64 **IPTCOffset** () const
- dng\_fingerprint **IPTCDigest** (bool includePadding=true) const
- void **RebuildIPTC** (bool padForTIFF, bool forceUTF8)
- bool **UsedUTF8forIPTC** () const
- void **SetUsedUTF8forIPTC** (bool used)
- bool **SetXMP** (dng\_host &host, const void \*buffer, uint32 count, bool xmpInSidecar=false, bool xmpIsNewer=false)
- dng\_xmp \* **GetXMP** ()
- const dng\_xmp \* **GetXMP** () const
- bool **XMPinSidecar** () const
- const dng\_linearization\_info \* **GetLinearizationInfo** () const

- void **ClearLinearizationInfo** ()
- void **SetLinearization** ([AutoPtr](#)< [dng\\_memory\\_block](#) > &curve)
- void **SetActiveArea** (const [dng\\_rect](#) &area)
- void **SetMaskedAreas** (uint32 count, const [dng\\_rect](#) \*area)
- void **SetMaskedArea** (const [dng\\_rect](#) &area)
- void **SetBlackLevel** (real64 black, int32 plane=-1)
- void **SetQuadBlacks** (real64 black0, real64 black1, real64 black2, real64 black3)
- void **SetRowBlacks** (const real64 \*blacks, uint32 count)
- void **SetColumnBlacks** (const real64 \*blacks, uint32 count)
- uint32 **WhiteLevel** (uint32 plane=0) const
- void **SetWhiteLevel** (uint32 white, int32 plane=-1)
- const [dng\\_mosaic\\_info](#) \* **GetMosaicInfo** () const
- void **ClearMosaicInfo** ()
- void **SetColorKeys** (ColorKeyCode color0, ColorKeyCode color1, ColorKeyCode color2, ColorKeyCode color3=colorKeyMaxEnum)
- void **SetRGB** ()
- void **SetCMY** ()
- void **SetGMCY** ()
- void **SetBayerMosaic** (uint32 phase)
- void **SetFujiMosaic** (uint32 phase)
- void **SetQuadMosaic** (uint32 pattern)
- void **SetGreenSplit** (uint32 split)
- const [dng\\_opcode\\_list](#) & **OpcodeList1** () const
- [dng\\_opcode\\_list](#) & **OpcodeList1** ()
- const [dng\\_opcode\\_list](#) & **OpcodeList2** () const
- [dng\\_opcode\\_list](#) & **OpcodeList2** ()
- const [dng\\_opcode\\_list](#) & **OpcodeList3** () const
- [dng\\_opcode\\_list](#) & **OpcodeList3** ()
- virtual void **Parse** ([dng\\_host](#) &host, [dng\\_stream](#) &stream, [dng\\_info](#) &info)
- virtual void **PostParse** ([dng\\_host](#) &host, [dng\\_stream](#) &stream, [dng\\_info](#) &info)
  
- virtual void **SynchronizeMetadata** ()
- void **UpdateDateTime** (const [dng\\_date\\_time\\_info](#) &dt)
- void **UpdateDateTimeToNow** ()
- virtual bool **SetFourColorBayer** ()
- const [dng\\_image](#) \* **Stage1Image** () const
- const [dng\\_image](#) \* **Stage2Image** () const
- const [dng\\_image](#) \* **Stage3Image** () const
- RawImageStageEnum **RawImageStage** () const
- const [dng\\_image](#) & **RawImage** () const
- virtual void **ReadStage1Image** ([dng\\_host](#) &host, [dng\\_stream](#) &stream, [dng\\_info](#) &info)

- void **SetStage1Image** ([AutoPtr](#)< [dng\\_image](#) > &image)
- void **SetStage2Image** ([AutoPtr](#)< [dng\\_image](#) > &image)
- void **SetStage3Image** ([AutoPtr](#)< [dng\\_image](#) > &image)
- void **BuildStage2Image** ([dng\\_host](#) &host, uint32 pixelType=ttShort)
- void **BuildStage3Image** ([dng\\_host](#) &host, int32 srcPlane=-1)
- void **SetStage3Gain** (real64 gain)
- real64 **Stage3Gain** () const
- void **SetIsPreview** (bool preview)
- bool **IsPreview** () const
- void **SetIsDamaged** (bool damaged)
- bool **IsDamaged** () const

#### Static Public Member Functions

- static [dng\\_negative](#) \* **Make** ([dng\\_memory\\_allocator](#) &allocator)

#### Protected Member Functions

- [dng\\_negative](#) ([dng\\_memory\\_allocator](#) &allocator)
- virtual void **Initialize** ()
- virtual [dng\\_exif](#) \* **MakeExif** ()
- virtual [dng\\_xmp](#) \* **MakeXMP** ()
- virtual [dng\\_linearization\\_info](#) \* **MakeLinearizationInfo** ()
- void **NeedLinearizationInfo** ()
- virtual [dng\\_mosaic\\_info](#) \* **MakeMosaicInfo** ()
- void **NeedMosaicInfo** ()
- virtual void **DoBuildStage2** ([dng\\_host](#) &host, uint32 pixelType)
- virtual void **DoInterpolateStage3** ([dng\\_host](#) &host, int32 srcPlane)
- virtual void **DoMergeStage3** ([dng\\_host](#) &host)
- virtual void **DoBuildStage3** ([dng\\_host](#) &host, int32 srcPlane)

#### Protected Attributes

- [dng\\_memory\\_allocator](#) & **fAllocator**
- [dng\\_string](#) **fModelName**
- [dng\\_string](#) **fLocalName**
- bool **fHasBaseOrientation**
- [dng\\_orientation](#) **fBaseOrientation**
- [dng\\_urational](#) **fDefaultCropSizeH**
- [dng\\_urational](#) **fDefaultCropSizeV**
- [dng\\_urational](#) **fDefaultCropOriginH**
- [dng\\_urational](#) **fDefaultCropOriginV**

- dng\_urational **fDefaultScaleH**
- dng\_urational **fDefaultScaleV**
- dng\_urational **fBestQualityScale**
- real64 **fRawToFullScaleH**
- real64 **fRawToFullScaleV**
- dng\_urational **fBaselineNoise**
- dng\_urational **fNoiseReductionApplied**
- [dng\\_noise\\_profile](#) **fNoiseProfile**
- dng\_srational **fBaselineExposure**
- dng\_urational **fBaselineSharpness**
- dng\_urational **fChromaBlurRadius**
- dng\_urational **fAntiAliasStrength**
- dng\_urational **fLinearResponseLimit**
- dng\_urational **fShadowScale**
- uint32 **fColorimetricReference**
- uint32 **fColorChannels**
- dng\_vector **fAnalogBalance**
- dng\_vector **fCameraNeutral**
- dng\_xy\_coord **fCameraWhiteXY**
- dng\_matrix **fCameraCalibration1**
- dng\_matrix **fCameraCalibration2**
- dng\_string **fCameraCalibrationSignature**
- std::vector< [dng\\_camera\\_profile](#) \* > **fCameraProfile**
- dng\_string **fAsShotProfileName**
- [dng\\_fingerprint](#) **fRawImageDigest**
- [dng\\_fingerprint](#) **fRawDataUniqueID**
- dng\_string **fOriginalRawFileName**
- bool **fHasOriginalRawFileData**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fOriginalRawFileData**
- [dng\\_fingerprint](#) **fOriginalRawFileDigest**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fDNGPrivateData**
- bool **fIsMakerNoteSafe**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fMakerNote**
- [AutoPtr](#)< [dng\\_exif](#) > **fExif**
- [AutoPtr](#)< [dng\\_exif](#) > **fOriginalExif**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fIPTCBlock**
- uint64 **fIPTCOffset**
- bool **fUsedUTF8forIPTC**
- [AutoPtr](#)< [dng\\_xmp](#) > **fXMP**
- bool **fValidEmbeddedXMP**
- bool **fXMPinSidecar**
- bool **fXMPisNewer**
- [AutoPtr](#)< [dng\\_linearization\\_info](#) > **fLinearizationInfo**

- [AutoPtr< dng\\_mosaic\\_info >](#) **fMosaicInfo**
- dng\_opcode\_list **fOpcodeList1**
- dng\_opcode\_list **fOpcodeList2**
- dng\_opcode\_list **fOpcodeList3**
- [AutoPtr< dng\\_image >](#) **fStage1Image**
- [AutoPtr< dng\\_image >](#) **fStage2Image**
- [AutoPtr< dng\\_image >](#) **fStage3Image**
- real64 **fStage3Gain**
- bool **fIsPreview**
- bool **fIsDamaged**
- RawImageStageEnum **fRawImageStage**
- [AutoPtr< dng\\_image >](#) **fRawImage**

### 6.40.1 Detailed Description

Main class for holding DNG image data and associated metadata.

### 6.40.2 Member Function Documentation

#### 6.40.2.1 void dng\_negative::ApplyOrientation (const dng\_orientation & orientation)

Logically rotates the image by changing the orientation values. This will also update the XMP data.

#### 6.40.2.2 uint32 dng\_negative::BestQualityFinalHeight () const [inline]

Get best quality height. For a naive conversion, one could use either the default size, or the best quality size.

References BestQualityScale(), DefaultScale(), and FinalHeight().

#### 6.40.2.3 uint32 dng\_negative::BestQualityFinalWidth () const [inline]

Get best quality width. For a naive conversion, one could use either the default size, or the best quality size.

References BestQualityScale(), DefaultScale(), and FinalWidth().

#### 6.40.2.4 `dng_rect dng_negative::DefaultCropArea (real64 scaleH = 1.0, real64 scaleV = 1.0) const`

The default crop area after applying the specified horizontal and vertical scale factors to the stage 3 image.

Referenced by `dng_renderer::Render()`.

#### 6.40.2.5 `real64 dng_negative::DefaultScale () const` `[inline]`

Get default scale factor. When specifying a single scale factor, we use the horizontal scale factor, and let the vertical scale factor be calculated based on the pixel aspect ratio.

References `DefaultScaleH()`.

Referenced by `BestQualityFinalHeight()`, `BestQualityFinalWidth()`, `DefaultFinalHeight()`, and `DefaultFinalWidth()`.

#### 6.40.2.6 `void dng_negative::SetCameraCalibration1 (const dng_matrix & m)`

Setter for first of up to two color matrices used for individual camera calibrations.

The sequence of matrix transforms is: Camera data  $\rightarrow$  camera calibration  $\rightarrow$  "inverse" of color matrix

This will be a 4x4 matrix for a four-color camera. The defaults are almost always the identity matrix, and for the cases where they aren't, they are diagonal matrices.

#### 6.40.2.7 `void dng_negative::SetCameraCalibration2 (const dng_matrix & m)`

Setter for second of up to two color matrices used for individual camera calibrations.

The sequence of matrix transforms is: Camera data  $\rightarrow$  camera calibration  $\rightarrow$  "inverse" of color matrix

This will be a 4x4 matrix for a four-color camera. The defaults are almost always the identity matrix, and for the cases where they aren't, they are diagonal matrices.

The documentation for this class was generated from the following files:

- [dng\\_negative.h](#)

- dng\_negative.cpp

## 6.41 dng\_noise\_function Class Reference

Noise model for photon and sensor read noise, assuming that they are independent random variables and spatially invariant.

```
#include <dng_negative.h>
```

Inheritance diagram for dng\_noise\_function::

### Public Member Functions

- **dng\_noise\_function** (real64 scale, real64 offset)
- virtual real64 [Evaluate](#) (real64 x) const
- real64 **Scale** () const
- real64 **Offset** () const
- void **SetScale** (real64 scale)
- void **SetOffset** (real64 offset)
- bool **IsValid** () const

### Protected Attributes

- real64 **fScale**
- real64 **fOffset**

#### 6.41.1 Detailed Description

Noise model for photon and sensor read noise, assuming that they are independent random variables and spatially invariant.

The noise model is  $N(x) = \sqrt{\text{scale} \cdot x + \text{offset}}$ , where  $x$  represents a linear signal value in the range  $[0,1]$ , and  $N(x)$  is the standard deviation (i.e., noise). The parameters scale and offset are both sensor-dependent and ISO-dependent. scale must be positive, and offset must be non-negative.

#### 6.41.2 Member Function Documentation

**6.41.2.1 virtual real64 dng\_noise\_function::Evaluate (real64 x) const**  
[inline, virtual]

Return the mapping for value  $x$ . This method must be implemented by a derived class of `dng_1d_function` and the derived class determines the lookup method and function used.

**Parameters:**

$x$  A value between 0.0 and 1.0 (inclusive).

**Return values:**

*Mapped* value for  $x$

Implements `dng_1d_function`.

The documentation for this class was generated from the following file:

- [dng\\_negative.h](#)

## 6.42 `dng_noise_profile` Class Reference

Noise profile for a negative.

```
#include <dng_negative.h>
```

**Public Member Functions**

- `dng_noise_profile` (const std::vector< [dng\\_noise\\_function](#) > &functions)
- `IsValid` () const
- `IsValidForNegative` (const [dng\\_negative](#) &negative) const
- `const dng_noise_function &NoiseFunction` (uint32 plane) const
- `uint32 NumFunctions` () const

**Protected Attributes**

- std::vector< [dng\\_noise\\_function](#) > `fNoiseFunctions`

### 6.42.1 Detailed Description

Noise profile for a negative.

For mosaiced negatives, the noise profile describes the approximate noise characteristics of a mosaic negative after linearization, but prior to demosaicing. For demosaiced negatives (i.e., linear DNGs), the noise profile describes the approximate noise characteristics of the image data immediately following the demosaic step, prior to the processing of opcode list 3.

A noise profile may contain 1 or N noise functions, where N is the number of color planes for the negative. Otherwise the noise profile is considered to be invalid for that negative. If the noise profile contains 1 noise function, then it is assumed that this single noise function applies to all color planes of the negative. Otherwise, the N noise functions map to the N planes of the negative in order specified in the `CFAPlaneColor` tag.

The documentation for this class was generated from the following files:

- [dng\\_negative.h](#)
- `dng_negative.cpp`

## 6.43 `dng_opcode_FixVignetteRadial` Class Reference

Radially-symmetric lens vignette correction opcode.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for `dng_opcode_FixVignetteRadial`:

### Public Member Functions

- `dng_opcode_FixVignetteRadial` (const [dng\\_vignette\\_radial\\_params](#) &params, uint32 flags)
- `dng_opcode_FixVignetteRadial` ([dng\\_stream](#) &stream)
- virtual bool `IsNOP` () const
- virtual bool `IsValidForNegative` (const [dng\\_negative](#) &) const
- virtual void `PutData` ([dng\\_stream](#) &stream) const
- virtual uint32 `BufferPixelType` (uint32)
- virtual void `Prepare` ([dng\\_negative](#) &negative, uint32 threadCount, const `dng_point` &tileSize, const `dng_rect` &imageBounds, uint32 imagePlanes, uint32 bufferPixelType, [dng\\_memory\\_allocator](#) &allocator)
- virtual void `ProcessArea` ([dng\\_negative](#) &negative, uint32 threadIndex, [dng\\_pixel\\_buffer](#) &buffer, const `dng_rect` &dstArea, const `dng_rect` &imageBounds)

### Static Protected Member Functions

- static uint32 `ParamBytes` ()

### Protected Attributes

- [dng\\_vignette\\_radial\\_params](#) **fParams**
- uint32 **fImagePlanes**
- int64 **fSrcOriginH**
- int64 **fSrcOriginV**
- int64 **fSrcStepH**
- int64 **fSrcStepV**
- uint32 **fTableInputBits**
- uint32 **fTableOutputBits**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fGainTable**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fMaskBuffers** [[kMaxMPThreads](#)]

#### 6.43.1 Detailed Description

Radially-symmetric lens vignette correction opcode.

The documentation for this class was generated from the following files:

- [dng\\_lens\\_correction.h](#)
- [dng\\_lens\\_correction.cpp](#)

## 6.44 dng\_opcode\_WarpFisheye Class Reference

Warp opcode for fisheye camera model.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for `dng_opcode_WarpFisheye::`

### Public Member Functions

- **dng\_opcode\_WarpFisheye** (const [dng\\_warp\\_params\\_fisheye](#) &params, uint32 flags)
- **dng\_opcode\_WarpFisheye** ([dng\\_stream](#) &stream)
- virtual bool **IsNOP** () const
- virtual bool **IsValidForNegative** (const [dng\\_negative](#) &negative) const
- virtual void **PutData** ([dng\\_stream](#) &stream) const
- virtual void **Apply** ([dng\\_host](#) &host, [dng\\_negative](#) &negative, [AutoPtr](#)< [dng\\_image](#) > &image)

### Static Protected Member Functions

- static uint32 **ParamBytes** (uint32 planes)

### Protected Attributes

- [dng\\_warp\\_params\\_fisheye](#) **fWarpParams**

#### 6.44.1 Detailed Description

Warp opcode for fisheye camera model.

The documentation for this class was generated from the following files:

- `dng_lens_correction.h`
- `dng_lens_correction.cpp`

## 6.45 `dng_opcode_WarpRectilinear` Class Reference

Warp opcode for pinhole perspective (rectilinear) camera model.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for `dng_opcode_WarpRectilinear`:

### Public Member Functions

- **dng\_opcode\_WarpRectilinear** (const [dng\\_warp\\_params\\_rectilinear](#) &params, uint32 flags)
- **dng\_opcode\_WarpRectilinear** ([dng\\_stream](#) &stream)
- virtual bool **IsNOP** () const
- virtual bool **IsValidForNegative** (const [dng\\_negative](#) &negative) const
- virtual void **PutData** ([dng\\_stream](#) &stream) const
- virtual void **Apply** ([dng\\_host](#) &host, [dng\\_negative](#) &negative, [AutoPtr](#)< [dng\\_image](#) > &image)

### Static Protected Member Functions

- static uint32 **ParamBytes** (uint32 planes)

**Protected Attributes**

- [dng\\_warp\\_params\\_rectilinear](#) `fWarpParams`

**6.45.1 Detailed Description**

Warp opcode for pinhole perspective (rectilinear) camera model.

The documentation for this class was generated from the following files:

- `dng_lens_correction.h`
- `dng_lens_correction.cpp`

**6.46 `dng_pixel_buffer` Class Reference**

Holds a buffer of pixel data with "pixel geometry" metadata.

```
#include <dng_pixel_buffer.h>
```

Inheritance diagram for `dng_pixel_buffer`:

**Public Member Functions**

- `dng_pixel_buffer` (const [dng\\_pixel\\_buffer](#) &buffer)
- `dng_pixel_buffer` & `operator=` (const [dng\\_pixel\\_buffer](#) &buffer)
- `uint32 PixelRange` () const
- const `dng_rect & Area` () const
- `uint32 Planes` () const
- `int32 RowStep` () const
- `int32 PlaneStep` () const
- const `void * ConstPixel` (int32 row, int32 col, uint32 plane=0) const
- `void * DirtyPixel` (int32 row, int32 col, uint32 plane=0)
- const `uint8 * ConstPixel_uint8` (int32 row, int32 col, uint32 plane=0) const
- `uint8 * DirtyPixel_uint8` (int32 row, int32 col, uint32 plane=0)
- const `int8 * ConstPixel_int8` (int32 row, int32 col, uint32 plane=0) const
- `int8 * DirtyPixel_int8` (int32 row, int32 col, uint32 plane=0)
- const `uint16 * ConstPixel_uint16` (int32 row, int32 col, uint32 plane=0) const
- `uint16 * DirtyPixel_uint16` (int32 row, int32 col, uint32 plane=0)
- const `int16 * ConstPixel_int16` (int32 row, int32 col, uint32 plane=0) const
- `int16 * DirtyPixel_int16` (int32 row, int32 col, uint32 plane=0)
- const `uint32 * ConstPixel_uint32` (int32 row, int32 col, uint32 plane=0) const
- `uint32 * DirtyPixel_uint32` (int32 row, int32 col, uint32 plane=0)

- `const int32 * ConstPixel_int32` (`int32 row, int32 col, uint32 plane=0`) `const`
- `int32 * DirtyPixel_int32` (`int32 row, int32 col, uint32 plane=0`)
- `const real32 * ConstPixel_real32` (`int32 row, int32 col, uint32 plane=0`) `const`
- `real32 * DirtyPixel_real32` (`int32 row, int32 col, uint32 plane=0`)
- `void SetConstant` (`const dng_rect &area, uint32 plane, uint32 planes, uint32 value`)
- `void SetConstant_uint8` (`const dng_rect &area, uint32 plane, uint32 planes, uint8 value`)
- `void SetConstant_uint16` (`const dng_rect &area, uint32 plane, uint32 planes, uint16 value`)
- `void SetConstant_int16` (`const dng_rect &area, uint32 plane, uint32 planes, int16 value`)
- `void SetConstant_uint32` (`const dng_rect &area, uint32 plane, uint32 planes, uint32 value`)
- `void SetConstant_real32` (`const dng_rect &area, uint32 plane, uint32 planes, real32 value`)
- `void SetZero` (`const dng_rect &area, uint32 plane, uint32 planes`)
- `void CopyArea` (`const dng_pixel_buffer &src, const dng_rect &area, uint32 srcPlane, uint32 dstPlane, uint32 planes`)
- `void CopyArea` (`const dng_pixel_buffer &src, const dng_rect &area, uint32 plane, uint32 planes`)
- `void RepeatArea` (`const dng_rect &srcArea, const dng_rect &dstArea`)
- `void RepeatSubArea` (`const dng_rect subArea, uint32 repeatV=1, uint32 repeatH=1`)

*Replicates a sub-area of a buffer to fill the entire buffer.*

- `void ShiftRight` (`uint32 shift`)
- `void FlipH` ()
- `void FlipV` ()
- `void FlipZ` ()
- `bool EqualArea` (`const dng_pixel_buffer &rhs, const dng_rect &area, uint32 plane, uint32 planes`) `const`
- `real64 MaximumDifference` (`const dng_pixel_buffer &rhs, const dng_rect &area, uint32 plane, uint32 planes`) `const`

#### Static Public Member Functions

- `static dng_point RepeatPhase` (`const dng_rect &srcArea, const dng_rect &dstArea`)

### Public Attributes

- dng\_rect **fArea**
- uint32 **fPlane**
- uint32 **fPlanes**
- int32 **fRowStep**
- int32 **fColStep**
- int32 **fPlaneStep**
- uint32 **fPixelType**
- uint32 **fPixelSize**
- void \* **fData**
- bool **fDirty**

#### 6.46.1 Detailed Description

Holds a buffer of pixel data with "pixel geometry" metadata.

The pixel geometry describes the layout in terms of how many planes, rows and columns plus the steps (in bytes) between each column, row and plane.

#### 6.46.2 Member Function Documentation

##### 6.46.2.1 const dng\_rect& dng\_pixel\_buffer::Area () const [inline]

Get extent of pixels in buffer

##### Return values:

*Rectangle* giving valid extent of buffer.

##### 6.46.2.2 const void\* dng\_pixel\_buffer::ConstPixel (int32 row, int32 col, uint32 plane = 0) const [inline]

Get read-only untyped (void \*) pointer to pixel data starting at a specific pixel in the buffer.

##### Parameters:

*row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as void \*.

Referenced by ConstPixel\_int16(), ConstPixel\_int32(), ConstPixel\_int8(), ConstPixel\_real32(), ConstPixel\_uint16(), ConstPixel\_uint32(), ConstPixel\_uint8(), CopyArea(), EqualArea(), MaximumDifference(), dng\_image::Put(), and RepeatArea().

**6.46.2.3** `const int16* dng_pixel_buffer::ConstPixel_int16 (int32 row, int32 col, uint32 plane = 0) const [inline]`

Get read-only int16 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

*row* Start row for buffer pointer.  
*col* Start column for buffer pointer.  
*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as int16 \*.

References ConstPixel().

**6.46.2.4** `const int32* dng_pixel_buffer::ConstPixel_int32 (int32 row, int32 col, uint32 plane = 0) const [inline]`

Get read-only int32 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

*row* Start row for buffer pointer.  
*col* Start column for buffer pointer.  
*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as int32 \*.

References ConstPixel().

### 6.46.2.5 `const int8* dng_pixel_buffer::ConstPixel_int8 (int32 row, int32 col, uint32 plane = 0) const` [inline]

Get read-only `int8 *` to pixel data starting at a specific pixel in the buffer.

#### Parameters:

- row* Start row for buffer pointer.
- col* Start column for buffer pointer.
- plane* Start plane for buffer pointer.

#### Return values:

*Pointer* to pixel data as `int8 *`.

References `ConstPixel()`.

### 6.46.2.6 `const real32* dng_pixel_buffer::ConstPixel_real32 (int32 row, int32 col, uint32 plane = 0) const` [inline]

Get read-only `real32 *` to pixel data starting at a specific pixel in the buffer.

#### Parameters:

- row* Start row for buffer pointer.
- col* Start column for buffer pointer.
- plane* Start plane for buffer pointer.

#### Return values:

*Pointer* to pixel data as `real32 *`.

References `ConstPixel()`.

### 6.46.2.7 `const uint16* dng_pixel_buffer::ConstPixel_uint16 (int32 row, int32 col, uint32 plane = 0) const` [inline]

Get read-only `uint16 *` to pixel data starting at a specific pixel in the buffer.

#### Parameters:

- row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as uint16 \*.

References ConstPixel().

**6.46.2.8** `const uint32* dng_pixel_buffer::ConstPixel_uint32 (int32 row, int32 col, uint32 plane = 0) const` [inline]

Get read-only uint32 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

*row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as uint32 \*.

References ConstPixel().

**6.46.2.9** `const uint8* dng_pixel_buffer::ConstPixel_uint8 (int32 row, int32 col, uint32 plane = 0) const` [inline]

Get read-only uint8 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

*row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as uint8 \*.

References ConstPixel().

**6.46.2.10** `void dng_pixel_buffer::CopyArea (const dng_pixel_buffer & src, const dng_rect & area, uint32 plane, uint32 planes) [inline]`

Copy image data from an area of one pixel buffer to same area of another.

**Parameters:**

- src* Buffer to copy from.
- area* Rectangle of pixel buffer to copy.
- plane* Plane to start copy in src and this.
- planes* Number of planes to copy.

References CopyArea().

**6.46.2.11** `void dng_pixel_buffer::CopyArea (const dng_pixel_buffer & src, const dng_rect & area, uint32 srcPlane, uint32 dstPlane, uint32 planes)`

Copy image data from an area of one pixel buffer to same area of another.

**Parameters:**

- src* Buffer to copy from.
- area* Rectangle of pixel buffer to copy.
- srcPlane* Plane to start copy in src.
- dstPlane* Plane to start copy in dst.
- planes* Number of planes to copy.

References ConstPixel(), DirtyPixel(), PixelRange(), and ThrowNotYetImplemented().

Referenced by CopyArea(), and dng\_image::CopyArea().

**6.46.2.12** `void* dng_pixel_buffer::DirtyPixel (int32 row, int32 col, uint32 plane = 0) [inline]`

Get a writable untyped (void \*) pointer to pixel data starting at a specific pixel in the buffer.

**Parameters:**

- row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as void \*.

References DNG\_ASSERT.

Referenced by CopyArea(), DirtyPixel\_int16(), DirtyPixel\_int32(), DirtyPixel\_int8(), DirtyPixel\_real32(), DirtyPixel\_uint16(), DirtyPixel\_uint32(), DirtyPixel\_uint8(), dng\_image::Get(), RepeatArea(), dng\_simple\_image::Rotate(), SetConstant(), ShiftRight(), and dng\_simple\_image::Trim().

**6.46.2.13** `int16* dng_pixel_buffer::DirtyPixel_int16 (int32 row, int32 col, uint32 plane = 0)` [inline]

Get a writable int16 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

*row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as int16 \*.

References DirtyPixel().

**6.46.2.14** `int32* dng_pixel_buffer::DirtyPixel_int32 (int32 row, int32 col, uint32 plane = 0)` [inline]

Get a writable int32 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

*row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as int32 \*.

References DirtyPixel().

**6.46.2.15 int8\* dng\_pixel\_buffer::DirtyPixel\_int8 (int32 row, int32 col, uint32 plane = 0) [inline]**

Get a writable int8 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

*row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as int8 \*.

References DirtyPixel().

**6.46.2.16 real32\* dng\_pixel\_buffer::DirtyPixel\_real32 (int32 row, int32 col, uint32 plane = 0) [inline]**

Get a writable real32 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

*row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as real32 \*.

References DirtyPixel().

**6.46.2.17** `uint16* dng_pixel_buffer::DirtyPixel_uint16 (int32 row, int32 col, uint32 plane = 0) [inline]`

Get a writable uint16 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

- row* Start row for buffer pointer.
- col* Start column for buffer pointer.
- plane* Start plane for buffer pointer.

**Return values:**

- Pointer* to pixel data as uint16 \*.

References DirtyPixel().

**6.46.2.18** `uint32* dng_pixel_buffer::DirtyPixel_uint32 (int32 row, int32 col, uint32 plane = 0) [inline]`

Get a writable uint32 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

- row* Start row for buffer pointer.
- col* Start column for buffer pointer.
- plane* Start plane for buffer pointer.

**Return values:**

- Pointer* to pixel data as uint32 \*.

References DirtyPixel().

**6.46.2.19** `uint8* dng_pixel_buffer::DirtyPixel_uint8 (int32 row, int32 col, uint32 plane = 0) [inline]`

Get a writable uint8 \* to pixel data starting at a specific pixel in the buffer.

**Parameters:**

- row* Start row for buffer pointer.

*col* Start column for buffer pointer.

*plane* Start plane for buffer pointer.

**Return values:**

*Pointer* to pixel data as uint8 \*.

References DirtyPixel().

**6.46.2.20 bool dng\_pixel\_buffer::EqualArea (const dng\_pixel\_buffer & rhs, const dng\_rect & area, uint32 plane, uint32 planes) const**

Return true if the contents of an area of the pixel buffer area are the same as those of another.

**Parameters:**

*rhs* Buffer to compare against.

*area* Rectangle of pixel buffer to test.

*plane* Plane to start comparing.

*planes* Number of planes to compare.

**Return values:**

*bool* true if areas are equal, false otherwise.

References ConstPixel(), and ThrowNotYetImplemented().

Referenced by dng\_image::EqualArea().

**6.46.2.21 void dng\_pixel\_buffer::FlipH ()**

Change metadata so pixels are iterated in opposite horizontal order. This operation does not require movement of actual pixel data.

**6.46.2.22 void dng\_pixel\_buffer::FlipV ()**

Change metadata so pixels are iterated in opposite vertical order. This operation does not require movement of actual pixel data.

### 6.46.2.23 void dng\_pixel\_buffer::FlipZ ()

Change metadata so pixels are iterated in opposite plane order. This operation does not require movement of actual pixel data.

### 6.46.2.24 real64 dng\_pixel\_buffer::MaximumDifference (const dng\_pixel\_buffer & rhs, const dng\_rect & area, uint32 plane, uint32 planes) const

Return the absolute value of the maximum difference between two pixel buffers. Used for comparison testing with tolerance

#### Parameters:

- rhs* Buffer to compare against.
- area* Rectangle of pixel buffer to test.
- plane* Plane to start comparing.
- planes* Number of planes to compare.

#### Return values:

- larges* absolute value difference between the corresponding pixels each buffer across area.

References ConstPixel(), ThrowNotYetImplemented(), and ThrowProgramError().

### 6.46.2.25 uint32 dng\_pixel\_buffer::PixelRange () const

Get the range of pixel values.

#### Return values:

- Range* of value a pixel can take. (Meaning [0, max] for unsigned case. Signed case is biased so [-32768, max - 32768].)

Referenced by CopyArea().

### 6.46.2.26 uint32 dng\_pixel\_buffer::Planes () const [inline]

Number of planes of image data.

**Return values:**

*Number* of planes held in buffer.

**6.46.2.27** `int32 dng_pixel_buffer::PlaneStep () const` `[inline]`

Step, in pixels not bytes, between planes of data in buffer.

**Return values:**

*plane* step in pixels. May be negative.

**6.46.2.28** `void dng_pixel_buffer::RepeatArea (const dng_rect & srcArea, const dng_rect & dstArea)`

Repeat the image data in `srcArea` across `dstArea`. (Generally used for padding operations.)

**Parameters:**

*srcArea* Area to repeat from.

*dstArea* Area to fill with data from `srcArea`.

References `ConstPixel()`, `DirtyPixel()`, `RepeatPhase()`, and `ThrowNotYetImplemented()`.

Referenced by `RepeatSubArea()`.

**6.46.2.29** `dng_point dng_pixel_buffer::RepeatPhase (const dng_rect & srcArea, const dng_rect & dstArea)` `[static]`

Calculate the offset phase of destination rectangle relative to source rectangle. Phase is based on a 0,0 origin and the notion of repeating `srcArea` across `dstArea`. It is the number of pixels into `srcArea` to start repeating from when tiling `dstArea`.

**Return values:**

*dng\_point* containing horizontal and vertical phase.

Referenced by `RepeatArea()`.

**6.46.2.30** `int32 dng_pixel_buffer::RowStep () const` `[inline]`

Step, in pixels not bytes, between rows of data in buffer.

**Return values:**

*row* step in pixels. May be negative.

**6.46.2.31** `void dng_pixel_buffer::SetConstant (const dng_rect & area, uint32 plane, uint32 planes, uint32 value)`

Initialize a rectangular area of pixel buffer to a constant.

**Parameters:**

*area* Rectangle of pixel buffer to set.

*plane* Plane to start filling on.

*planes* Number of planes to fill.

*value* Constant value to set pixels to.

References `DirtyPixel()`, and `ThrowNotYetImplemented()`.

Referenced by `SetConstant_int16()`, `SetConstant_real32()`, `SetConstant_uint16()`, `SetConstant_uint32()`, `SetConstant_uint8()`, and `SetZero()`.

**6.46.2.32** `void dng_pixel_buffer::SetConstant_int16 (const dng_rect & area, uint32 plane, uint32 planes, int16 value)` `[inline]`

Initialize a rectangular area of pixel buffer to a constant signed 16-bit value.

**Parameters:**

*area* Rectangle of pixel buffer to set.

*plane* Plane to start filling on.

*planes* Number of planes to fill.

*value* Constant int16 value to set pixels to.

References `DNG_ASSERT`, and `SetConstant()`.

**6.46.2.33** `void dng_pixel_buffer::SetConstant_real32 (const dng_rect & area, uint32 plane, uint32 planes, real32 value) [inline]`

Initialize a rectangular area of pixel buffer to a constant real 32-bit value.

**Parameters:**

- area* Rectangle of pixel buffer to set.
- plane* Plane to start filling on.
- planes* Number of planes to fill.
- value* Constant real32 value to set pixels to.

References DNG\_ASSERT, and SetConstant().

**6.46.2.34** `void dng_pixel_buffer::SetConstant_uint16 (const dng_rect & area, uint32 plane, uint32 planes, uint16 value) [inline]`

Initialize a rectangular area of pixel buffer to a constant unsigned 16-bit value.

**Parameters:**

- area* Rectangle of pixel buffer to set.
- plane* Plane to start filling on.
- planes* Number of planes to fill.
- value* Constant uint16 value to set pixels to.

References DNG\_ASSERT, and SetConstant().

**6.46.2.35** `void dng_pixel_buffer::SetConstant_uint32 (const dng_rect & area, uint32 plane, uint32 planes, uint32 value) [inline]`

Initialize a rectangular area of pixel buffer to a constant unsigned 32-bit value.

**Parameters:**

- area* Rectangle of pixel buffer to set.
- plane* Plane to start filling on.
- planes* Number of planes to fill.
- value* Constant uint32 value to set pixels to.

References DNG\_ASSERT, and SetConstant().

**6.46.2.36** `void dng_pixel_buffer::SetConstant_uint8 (const dng_rect & area,  
uint32 plane, uint32 planes, uint8 value)` [`inline`]

Initialize a rectangular area of pixel buffer to a constant unsigned 8-bit value.

**Parameters:**

- area* Rectangle of pixel buffer to set.
- plane* Plane to start filling on.
- planes* Number of planes to fill.
- value* Constant uint8 value to set pixels to.

References `DNG_ASSERT`, and `SetConstant()`.

**6.46.2.37** `void dng_pixel_buffer::SetZero (const dng_rect & area, uint32 plane,  
uint32 planes)`

Initialize a rectangular area of pixel buffer to zeros.

**Parameters:**

- area* Rectangle of pixel buffer to zero.
- area* Area to zero
- plane* Plane to start filling on.
- planes* Number of planes to fill.

References `SetConstant()`, and `ThrowNotYetImplemented()`.

**6.46.2.38** `void dng_pixel_buffer::ShiftRight (uint32 shift)`

Apply a right shift (C++ oerpator `>>`) to all pixel values. Only implemented for 16-bit (signed or unsigned) pixel buffers.

**Parameters:**

- shift* Number of bits by which to right shift each pixel value.

References `DirtyPixel()`, and `ThrowNotYetImplemented()`.

The documentation for this class was generated from the following files:

- [dng\\_pixel\\_buffer.h](#)
- [dng\\_pixel\\_buffer.cpp](#)

## 6.47 `dng_render` Class Reference

Class used to render digital negative to displayable image.

```
#include <dng_render.h>
```

### Public Member Functions

- [dng\\_render](#) ([dng\\_host](#) &host, const [dng\\_negative](#) &negative)
- void [SetWhiteXY](#) (const [dng\\_xy\\_coord](#) &white)
- const [dng\\_xy\\_coord](#) [WhiteXY](#) () const
- void [SetExposure](#) (real64 exposure)
- real64 [Exposure](#) () const
- void [SetShadows](#) (real64 shadows)
- real64 [Shadows](#) () const
- void [SetToneCurve](#) (const [dng\\_1d\\_function](#) &curve)
- const [dng\\_1d\\_function](#) & [ToneCurve](#) () const
- void [SetFinalSpace](#) (const [dng\\_color\\_space](#) &space)
- const [dng\\_color\\_space](#) & [FinalSpace](#) () const
- void [SetFinalPixelFormat](#) (uint32 type)
- uint32 [FinalPixelFormat](#) () const
- void [SetMaximumSize](#) (uint32 size)
- uint32 [MaximumSize](#) () const
- virtual [dng\\_image](#) \* [Render](#) ()

### Protected Attributes

- [dng\\_host](#) & **fHost**
- const [dng\\_negative](#) & **fNegative**
- [dng\\_xy\\_coord](#) **fWhiteXY**
- real64 **fExposure**
- real64 **fShadows**
- const [dng\\_1d\\_function](#) \* **fToneCurve**
- const [dng\\_color\\_space](#) \* **fFinalSpace**
- uint32 **fFinalPixelFormat**
- uint32 **fMaximumSize**

#### 6.47.1 Detailed Description

Class used to render digital negative to displayable image.

## 6.47.2 Constructor & Destructor Documentation

### 6.47.2.1 `dng_render::dng_render (dng_host & host, const dng_negative & negative)`

Construct a rendering instance that will be used to convert a given digital negative.

#### Parameters:

*host* The host to use for memory allocation, progress updates, and abort testing.

*negative* The digital negative to convert to a displayable image.

References `AutoPtr< T >::Get()`, `dng_1d_identity::Get()`, and `AutoPtr< T >::Reset()`.

## 6.47.3 Member Function Documentation

### 6.47.3.1 `real64 dng_render::Exposure () const` `[inline]`

Get exposure compensation.

#### Return values:

*Compensation* value in stops, positive or negative.

### 6.47.3.2 `uint32 dng_render::FinalPixelType () const` `[inline]`

Get pixel type of final image data. Can be `ttByte` (default), `ttShort`, or `ttFloat`.

#### Return values:

*Pixel* type to use.

Referenced by `Render()`.

### 6.47.3.3 `const dng_color_space& dng_render::FinalSpace () const` `[inline]`

Get final color space in which resulting image data should be represented.

**Return values:**

*Color* space to use.

Referenced by `Render()`.

**6.47.3.4** `uint32 dng_render::MaximumSize () const` [inline]

Get maximum dimension, in pixels, of resulting image. If the final image would have either dimension larger than this maximum, the larger of the two dimensions is set to this maximum size and the smaller dimension is adjusted to preserve the image's aspect ratio.

**Return values:**

*Maximum* allowed size.

Referenced by `Render()`.

**6.47.3.5** `dng_image * dng_render::Render ()` [virtual]

Actually render a digital negative to a displayable image. Input digital negative is passed to the constructor of this `dng_render` class.

**Return values:**

*The* final resulting image.

References `dng_negative::AspectRatio()`, `dng_negative::DefaultCropArea()`, `dng_negative::DefaultFinalHeight()`, `dng_negative::DefaultFinalWidth()`, `FinalPixelType()`, `FinalSpace()`, `AutoPtr< T >::Get()`, `dng_color_space::IsMonochrome()`, `dng_host::Make_dng_image()`, `MaximumSize()`, `dng_host::PerformAreaTask()`, `dng_image::PixelType()`, `dng_image::Planes()`, `AutoPtr< T >::Release()`, and `AutoPtr< T >::Reset()`.

**6.47.3.6** `void dng_render::SetExposure (real64 exposure)` [inline]

Set exposure compensation.

**Parameters:**

*exposure* Compensation value in stops, positive or negative.

**6.47.3.7** `void dng_render::SetFinalPixelType (uint32 type)` [inline]

Set pixel type of final image data. Can be `ttByte` (default), `ttShort`, or `ttFloat`.

**Parameters:**

*type* Pixel type to use.

**6.47.3.8** `void dng_render::SetFinalSpace (const dng_color_space & space)`  
[inline]

Set final color space in which resulting image data should be represented. (See [dng\\_color\\_space.h](#) for possible values.)

**Parameters:**

*space* Color space to use.

**6.47.3.9** `void dng_render::SetMaximumSize (uint32 size)` [inline]

Set maximum dimension, in pixels, of resulting image. If final image would have either dimension larger than maximum, the larger of the two dimensions is set to this maximum size and the smaller dimension is adjusted to preserve aspect ratio.

**Parameters:**

*size* Maximum size to allow.

**6.47.3.10** `void dng_render::SetShadows (real64 shadows)` [inline]

Set shadow clip amount.

**Parameters:**

*shadows* Shadow clip amount.

**6.47.3.11** `void dng_render::SetToneCurve (const dng_1d_function & curve)`  
[inline]

Set custom tone curve for conversion.

**Parameters:**

*curve* 1D function that defines tone mapping to use during conversion.

**6.47.3.12** `void dng_render::SetWhiteXY (const dng_xy_coord & white)`  
[inline]

Set the white point to be used for conversion.

**Parameters:**

*white* White point to use.

**6.47.3.13** `real64 dng_render::Shadows () const` [inline]

Get shadow clip amount.

**Return values:**

*Shadow* clip amount.

**6.47.3.14** `const dng_1d_function& dng_render::ToneCurve () const`  
[inline]

Get custom tone curve for conversion.

**Return values:**

*ID* function that defines tone mapping to use during conversion.

### 6.47.3.15 `const dng_xy_coord dng_render::WhiteXY () const` [inline]

Get the white point to be used for conversion.

#### Return values:

*White* point to use.

The documentation for this class was generated from the following files:

- [dng\\_render.h](#)
- [dng\\_render.cpp](#)

## 6.48 `dng_simple_image` Class Reference

[dng\\_image](#) derived class with simple Trim and Rotate functionality.

```
#include <dng_simple_image.h>
```

Inheritance diagram for `dng_simple_image::`

### Public Member Functions

- `dng_simple_image` (const `dng_rect` &bounds, uint32 planes, uint32 pixelType, [dng\\_memory\\_allocator](#) &allocator)
- virtual `dng_image * Clone () const`
- virtual void [SetPixelType](#) (uint32 pixelType)  
*Setter for pixel type.*
- virtual void [Trim](#) (const `dng_rect` &r)  
*Trim image data outside of given bounds. Memory is not reallocated or freed.*
- virtual void [Rotate](#) (const `dng_orientation` &orientation)  
*Rotate image according to orientation.*
- void [GetPixelBuffer](#) (`dng_pixel_buffer` &buffer)  
*Get the buffer for direct processing. (Unique to [dng\\_simple\\_image](#).)*

### Protected Member Functions

- virtual void **AcquireTileBuffer** ([dng\\_tile\\_buffer](#) &buffer, const [dng\\_rect](#) &area, bool dirty) const

### Protected Attributes

- [dng\\_pixel\\_buffer](#) **fBuffer**
- [AutoPtr](#)< [dng\\_memory\\_block](#) > **fMemory**
- [dng\\_memory\\_allocator](#) & **fAllocator**

#### 6.48.1 Detailed Description

[dng\\_image](#) derived class with simple Trim and Rotate functionality.

The documentation for this class was generated from the following files:

- [dng\\_simple\\_image.h](#)
- [dng\\_simple\\_image.cpp](#)

## 6.49 dng\_sniffer\_task Class Reference

Class to establish scope of a named subtask in DNG processing.

```
#include <dng_abort_sniffer.h>
```

### Public Member Functions

- [dng\\_sniffer\\_task](#) ([dng\\_abort\\_sniffer](#) \*sniffer, const char \*name=NULL, real64 fract=0.0)
- void [Sniff](#) ()
- void [UpdateProgress](#) (real64 fract)
- void [UpdateProgress](#) (uint32 done, uint32 total)
- void [Finish](#) ()

*Signal task completed for progress purposes.*

#### 6.49.1 Detailed Description

Class to establish scope of a named subtask in DNG processing.

Instances of this class are intended to be stack allocated.

## 6.49.2 Constructor & Destructor Documentation

### 6.49.2.1 `dng_sniffer_task::dng_sniffer_task (dng_abort_sniffer * sniffer, const char * name = NULL, real64 fract = 0.0)` [inline]

Inform a sniffer of a subtask in DNG processing.

#### Parameters:

*sniffer* The sniffer associated with the host on which this processing is occurring.

*name* The name of this subtask as a NUL terminated string.

*fract* Percentage of total processing this task is expected to take, from 0.0 to 1.0 .

References `dng_abort_sniffer::StartTask()`.

## 6.49.3 Member Function Documentation

### 6.49.3.1 `void dng_sniffer_task::Sniff ()` [inline]

Check for pending user cancellation or other abort. `ThrowUserCanceled` will be called if one is pending.

References `dng_abort_sniffer::SniffForAbort()`.

### 6.49.3.2 `void dng_sniffer_task::UpdateProgress (uint32 done, uint32 total)` [inline]

Update progress on this subtask.

#### Parameters:

*done* Amount of task completed in arbitrary integer units.

*total* Total size of task in same arbitrary integer units as done.

References `UpdateProgress()`.

### 6.49.3.3 `void dng_sniffer_task::UpdateProgress (real64 fract)` [inline]

Update progress on this subtask.

**Parameters:**

*fract* Percentage of processing completed on current task, from 0.0 to 1.0 .

References `dng_abort_sniffer::UpdateProgress()`.

Referenced by `Finish()`, and `UpdateProgress()`.

The documentation for this class was generated from the following file:

- [dng\\_abort\\_sniffer.h](#)

## 6.50 `dng_space_AdobeRGB` Class Reference

Singleton class for AdobeRGB color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for `dng_space_AdobeRGB::`

**Public Member Functions**

- virtual const [dng\\_1d\\_function](#) & [GammaFunction](#) () const  
*Returns `dng_function_GammaEncode_1_8`.*
- virtual bool [ICCProfile](#) (uint32 &size, const uint8 \*&data) const  
*Returns AdobeRGB (1998) ICC profile.*

**Static Public Member Functions**

- static const [dng\\_color\\_space](#) & [Get](#) ()  
*Static method for getting single global instance of this color space.*

### 6.50.1 Detailed Description

Singleton class for AdobeRGB color space.

The documentation for this class was generated from the following files:

- [dng\\_color\\_space.h](#)
- [dng\\_color\\_space.cpp](#)

## 6.51 dng\_space\_ColorMatch Class Reference

Singleton class for ColorMatch color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng\_space\_ColorMatch::

### Public Member Functions

- virtual const [dng\\_1d\\_function](#) & [GammaFunction](#) () const  
*Returns [dng\\_function\\_GammaEncode\\_1\\_8](#).*
- virtual bool [ICCProfile](#) (uint32 &size, const uint8 \*&data) const  
*Returns ColorMatch RGB ICC profile.*

### Static Public Member Functions

- static const [dng\\_color\\_space](#) & [Get](#) ()  
*Static method for getting single global instance of this color space.*

#### 6.51.1 Detailed Description

Singleton class for ColorMatch color space.

The documentation for this class was generated from the following files:

- [dng\\_color\\_space.h](#)
- [dng\\_color\\_space.cpp](#)

## 6.52 dng\_space\_GrayGamma18 Class Reference

Singleton class for gamma 1.8 grayscale color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for dng\_space\_GrayGamma18::

### Public Member Functions

- virtual const `dng_1d_function & GammaFunction ()` const  
*Returns `dng_function_GammaEncode_1_8`.*
- virtual bool `ICCProfile (uint32 &size, const uint8 *&data)` const  
*Returns simple grayscale gamma 1.8 ICC profile.*

### Static Public Member Functions

- static const `dng_color_space & Get ()`  
*Static method for getting single global instance of this color space.*

#### 6.52.1 Detailed Description

Singleton class for gamma 1.8 grayscale color space.

The documentation for this class was generated from the following files:

- `dng_color_space.h`
- `dng_color_space.cpp`

### 6.53 `dng_space_GrayGamma22` Class Reference

Singleton class for gamma 2.2 grayscale color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for `dng_space_GrayGamma22::`

### Public Member Functions

- virtual const `dng_1d_function & GammaFunction ()` const  
*Returns `dng_function_GammaEncode_2_2`.*
- virtual bool `ICCProfile (uint32 &size, const uint8 *&data)` const  
*Returns simple grayscale gamma 2.2 ICC profile.*

### Static Public Member Functions

- static const [dng\\_color\\_space](#) & `Get ()`  
*Static method for getting single global instance of this color space.*

#### 6.53.1 Detailed Description

Singleton class for gamma 2.2 grayscale color space.

The documentation for this class was generated from the following files:

- [dng\\_color\\_space.h](#)
- `dng_color_space.cpp`

## 6.54 `dng_space_ProPhoto` Class Reference

Singleton class for ProPhoto RGB color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for `dng_space_ProPhoto`::

### Public Member Functions

- virtual const [dng\\_1d\\_function](#) & `GammaFunction ()` const  
*Returns `dng_function_GammaEncode_1_8`.*
- virtual bool `ICCProfile (uint32 &size, const uint8 *&data)` const  
*Returns ProPhoto RGB ICC profile.*

### Static Public Member Functions

- static const [dng\\_color\\_space](#) & `Get ()`  
*Static method for getting single global instance of this color space.*

### 6.54.1 Detailed Description

Singleton class for ProPhoto RGB color space.

The documentation for this class was generated from the following files:

- [dng\\_color\\_space.h](#)
- [dng\\_color\\_space.cpp](#)

## 6.55 `dng_space_sRGB` Class Reference

Singleton class for sRGB color space.

```
#include <dng_color_space.h>
```

Inheritance diagram for `dng_space_sRGB`:

### Public Member Functions

- virtual const [dng\\_1d\\_function](#) & [GammaFunction](#) () const  
*Returns `dng_function_GammaEncode_sRGB`.*
- virtual bool [ICCProfile](#) (uint32 &size, const uint8 \*&data) const  
*Returns `sRGB IEC61966-2.1 ICC profile`.*

### Static Public Member Functions

- static const [dng\\_color\\_space](#) & [Get](#) ()  
*Static method for getting single global instance of this color space.*

### 6.55.1 Detailed Description

Singleton class for sRGB color space.

The documentation for this class was generated from the following files:

- [dng\\_color\\_space.h](#)
- [dng\\_color\\_space.cpp](#)

## 6.56 dng\_stream Class Reference

```
#include <dng_stream.h>
```

Inheritance diagram for dng\_stream::

### Public Types

- enum { **kSmallBufferSize** = 4 \* 1024, **kBigBufferSize** = 64 \* 1024, **kDefaultBufferSize** = kSmallBufferSize }

### Public Member Functions

- [dng\\_stream](#) (const void \*data, uint32 count, uint64 offsetInOriginalFile=kDNGStreamInvalidOffset)
- bool [SwapBytes](#) () const
- void [SetSwapBytes](#) (bool swapBytes)
- bool [BigEndian](#) () const
- void [SetBigEndian](#) (bool bigEndian=true)
- bool [LittleEndian](#) () const
- void [SetLittleEndian](#) (bool littleEndian=true)
- uint32 [BufferSize](#) () const

*Returns the size of the buffer used by the stream.*

- uint64 [Length](#) ()
- uint64 [Position](#) () const
- uint64 [PositionInOriginalFile](#) () const
- uint64 [OffsetInOriginalFile](#) () const
- const void \* [Data](#) () const
- [dng\\_memory\\_block](#) \* [AsMemoryBlock](#) ([dng\\_memory\\_allocator](#) &allocator)
- void [SetReadPosition](#) (uint64 offset)

*Seek to a new position in stream for reading.*

- void [Skip](#) (uint64 delta)
- void [Get](#) (void \*data, uint32 count)
- void [SetWritePosition](#) (uint64 offset)

*Seek to a new position in stream for writing.*

- void [Flush](#) ()

*Force any stored data in stream to be written to underlying storage.*

- void `SetLength` (uint64 length)
- void `Put` (const void \*data, uint32 count)
- uint8 `Get_uint8` ()
- void `Put_uint8` (uint8 x)
- uint16 `Get_uint16` ()
- void `Put_uint16` (uint16 x)
- uint32 `Get_uint32` ()
- void `Put_uint32` (uint32 x)
- uint64 `Get_uint64` ()
- void `Put_uint64` (uint64 x)
- int8 `Get_int8` ()
- void `Put_int8` (int8 x)
- int16 `Get_int16` ()
- void `Put_int16` (int16 x)
- int32 `Get_int32` ()
- void `Put_int32` (int32 x)
- int64 `Get_int64` ()
- void `Put_int64` (int64 x)
- real32 `Get_real32` ()
- void `Put_real32` (real32 x)
- real64 `Get_real64` ()
- void `Put_real64` (real64 x)
- void `Get_CString` (char \*data, uint32 maxLength)
- void `Get_UString` (char \*data, uint32 maxLength)
- void `PutZeros` (uint64 count)
- void `PadAlign2` ()  
*Writes zeros to align the stream position to a multiple of 2.*
- void `PadAlign4` ()  
*Writes zeros to align the stream position to a multiple of 4.*
- uint32 `TagValue_uint32` (uint32 tagType)
- int32 `TagValue_int32` (uint32 tagType)
- `dng_urational` `TagValue_urational` (uint32 tagType)
- `dng_srational` `TagValue_srational` (uint32 tagType)
- real64 `TagValue_real64` (uint32 tagType)
- `dng_abort_sniffer` \* `Sniffer` () const
- void `SetSniffer` (`dng_abort_sniffer` \*sniffer)
- virtual void `CopyToStream` (`dng_stream` &dstStream, uint64 count)
- void `DuplicateStream` (`dng_stream` &dstStream)

### Protected Member Functions

- `dng_stream` (`dng_abort_sniffer` `*sniffer=NULL`, `uint32`  
`bufferSize=kDefaultBufferSize`, `uint64` `offsetInOriginal-`  
`File=kDNGStreamInvalidOffset`)
- virtual `uint64 DoGetLength` ()
- virtual void `DoRead` (`void *data`, `uint32 count`, `uint64 offset`)
- virtual void `DoSetLength` (`uint64 length`)
- virtual void `DoWrite` (`const void *data`, `uint32 count`, `uint64 offset`)

#### 6.56.1 Detailed Description

Base stream abstraction. Has support for going between stream and pointer abstraction.

#### 6.56.2 Constructor & Destructor Documentation

##### 6.56.2.1 `dng_stream::dng_stream` (`const void *data`, `uint32 count`, `uint64` `offsetInOriginalFile = kDNGStreamInvalidOffset`)

Construct a stream with initial data.

#### Parameters:

*data* Pointer to initial contents of stream.

*count* Number of bytes data is valid for.

*offsetInOriginalFile* If data came from a file originally, offset can be saved here for later use.

#### 6.56.3 Member Function Documentation

##### 6.56.3.1 `dng_memory_block * dng_stream::AsMemoryBlock` (`dng_memory_allocator & allocator`)

Return the entire stream as a single memory block. This works for all streams, but requires copying the data to a new buffer.

#### Parameters:

*allocator* Allocator used to allocate memory.

References `dng_memory_allocator::Allocate()`, `Flush()`, `Get()`, `Length()`, `SetReadPosition()`, and `ThrowProgramError()`.

Referenced by `dng_iptc::Spool()`.

### 6.56.3.2 `bool dng_stream::BigEndian () const`

Getter for whether data in stream is big endian.

#### Return values:

*If* true, data in stream is big endian.

Referenced by `LittleEndian()`, `dng_image_writer::WriteDNG()`, and `dng_image_writer::WriteTIFFWithProfile()`.

### 6.56.3.3 `void dng_stream::CopyToStream (dng_stream & dstStream, uint64 count) [virtual]`

Copy a specified number of bytes to a target stream.

#### Parameters:

*dstStream* The target stream.

*count* The number of bytes to copy.

Reimplemented in [dng\\_memory\\_stream](#).

References `dng_memory_data::Buffer()`, `Get()`, and `Put()`.

Referenced by `DuplicateStream()`.

### 6.56.3.4 `const void * dng_stream::Data () const`

Return pointer to stream contents if the stream is entirely available as a single memory block, NULL otherwise.

### 6.56.3.5 `void dng_stream::DuplicateStream (dng_stream & dstStream)`

Makes the target stream a copy of this stream.

**Parameters:**

*dstStream* The target stream.

References `CopyToStream()`, `Flush()`, `Length()`, `SetLength()`, `SetReadPosition()`, and `SetWritePosition()`.

**6.56.3.6 void `dng_stream::Get` (void \* *data*, uint32 *count*)**

Get data from stream. Exception is thrown and no data is read if insufficient data available in stream.

**Parameters:**

*data* Buffer to put data into. Must be valid for count bytes.

*count* Bytes of data to read.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Flush()`, `Length()`, `dng_abort_sniffer::SniffForAbort()`, and `ThrowEndOfFile()`.

Referenced by `AsMemoryBlock()`, `CopyToStream()`, `Get_real64()`, `Get_uint16()`, `Get_uint32()`, `Get_uint64()`, `Get_uint8()`, and `dng_iptc::Parse()`.

**6.56.3.7 void `dng_stream::Get_CString` (char \* *data*, uint32 *maxLength*)**

Get an 8-bit character string from stream and advance read position. Routine always reads until a NUL character (8-bits of zero) is read. (That is, only `maxLength` bytes will be returned in buffer, but the stream is always advanced until a NUL is read or EOF is reached.)

**Parameters:**

*data* Buffer in which string is returned.

*maxLength* Maximum number of bytes to place in buffer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if stream runs out before NUL is seen.

References `Get_uint8()`.

### 6.56.3.8 int16 dng\_stream::Get\_int16 () [inline]

Get one 16-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

**Return values:**

*One* 16-bit integer.

**Exceptions:**

*dng\_exception* with fErrorCode equal to dng\_error\_end\_of\_file if not enough data in stream.

References Get\_uint16().

Referenced by TagValue\_int32().

### 6.56.3.9 int32 dng\_stream::Get\_int32 () [inline]

Get one 32-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

**Return values:**

*One* 32-bit integer.

**Exceptions:**

*dng\_exception* with fErrorCode equal to dng\_error\_end\_of\_file if not enough data in stream.

References Get\_uint32().

Referenced by TagValue\_int32(), TagValue\_real64(), TagValue\_srational(), and TagValue\_urational().

### 6.56.3.10 int64 dng\_stream::Get\_int64 () [inline]

Get one 64-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

**Return values:**

*One* 64-bit integer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get_uint64()`.

**6.56.3.11** `int8 dng_stream::Get_int8 ()` [inline]

Get one 8-bit integer from stream and advance read position.

**Return values:**

*One* 8-bit integer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get_uint8()`.

Referenced by `dng_iptc::Parse()`, and `TagValue_int32()`.

**6.56.3.12** `real32 dng_stream::Get_real32 ()`

Get one 32-bit IEEE floating-point number from stream and advance read position. Byte swap if byte swapping is turned on.

**Return values:**

*One* 32-bit IEEE floating-point number.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get_uint32()`.

Referenced by `dng_camera_profile::Parse()`, and `TagValue_real64()`.

### 6.56.3.13 `real64 dng_stream::Get_real64 ()`

Get one 64-bit IEEE floating-point number from stream and advance read position. Byte swap if byte swapping is turned on.

**Return values:**

*One* 64-bit IEEE floating-point number .

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get()`, and `Get_uint32()`.

Referenced by `TagValue_real64()`.

### 6.56.3.14 `uint16 dng_stream::Get_uint16 ()`

Get an unsigned 16-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

**Return values:**

*One* unsigned 16-bit integer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get()`.

Referenced by `Get_int16()`, `Get_UString()`, `dng_iptc::Parse()`, `dng_info::Parse()`, and `TagValue_uint32()`.

### 6.56.3.15 `uint32 dng_stream::Get_uint32 ()`

Get an unsigned 32-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

**Return values:**

*One* unsigned 32-bit integer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get()`.

Referenced by `Get_int32()`, `Get_real32()`, `Get_real64()`, `Get_uint64()`, `dng_info::Parse()`, `TagValue_real64()`, `TagValue_uint32()`, and `TagValue_urational()`.

**6.56.3.16 `uint64 dng_stream::Get_uint64 ()`**

Get an unsigned 64-bit integer from stream and advance read position. Byte swap if byte swapping is turned on.

**Return values:**

*One* unsigned 64-bit integer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get()`, and `Get_uint32()`.

Referenced by `Get_int64()`.

**6.56.3.17 `uint8 dng_stream::Get_uint8 ()` `[inline]`**

Get an unsigned 8-bit integer from stream and advance read position.

**Return values:**

*One* unsigned 8-bit integer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References Get().

Referenced by Get\_CString(), Get\_int8(), dng\_iptc::Parse(), and TagValue\_uint32().

#### 6.56.3.18 void dng\_stream::Get\_UString (char \* *data*, uint32 *maxLength*)

Get a 16-bit character string from stream and advance read position. 16-bit characters are truncated to 8-bits. Routine always reads until a NUL character (16-bits of zero) is read. (That is, only *maxLength* bytes will be returned in buffer, but the stream is always advanced until a NUL is read or EOF is reached.)

##### Parameters:

*data* Buffer to place string in.

*maxLength* Maximum number of bytes to place in buffer.

##### Exceptions:

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if stream runs out before NUL is seen.

References Get\_uint16().

#### 6.56.3.19 uint64 dng\_stream::Length () [inline]

Getter for length of data in stream.

##### Return values:

*Length* of readable data in stream.

Referenced by AsMemoryBlock(), dng\_memory\_stream::CopyToStream(), DuplicateStream(), Get(), dng\_iptc::Parse(), dng\_info::Parse(), Put(), Put\_uint8(), SetLength(), SetReadPosition(), dng\_iptc::Spool(), dng\_image\_writer::WriteDNG(), and dng\_image\_writer::WriteTIFFWithProfile().

#### 6.56.3.20 bool dng\_stream::LittleEndian () const [inline]

Getter for whether data in stream is big endian.

**Return values:**

*If* true, data in stream is big endian.

References BigEndian().

**6.56.3.21 uint64 dng\_stream::OffsetInOriginalFile () const**

Getter for offset in original file.

**Return values:**

*kInvalidOffset* if no offset in original file is set, offset in original file otherwise.

**6.56.3.22 uint64 dng\_stream::Position () const [inline]**

Getter for current offset in stream.

**Return values:**

*current* offset from start of stream.

Referenced by dng\_memory\_stream::CopyToStream(), PadAlign2(), PadAlign4(), dng\_iptc::Parse(), dng\_info::Parse(), PositionInOriginalFile(), Skip(), dng\_image\_writer::WriteDNG(), and dng\_image\_writer::WriteTIFFWithProfile().

**6.56.3.23 uint64 dng\_stream::PositionInOriginalFile () const**

Getter for current position in original file, taking into account OffsetInOriginalFile stream data was taken from.

**Return values:**

*kInvalidOffset* if no offset in original file is set, sum of offset in original file and current position otherwise.

References Position().

Referenced by dng\_info::Parse().

**6.56.3.24 void dng\_stream::Put (const void \* *data*, uint32 *count*)**

Write data to stream.

**Parameters:**

*data* Buffer of data to write to stream.

*count* Bytes of in data.

References Flush(), Length(), and dng\_abort\_sniffer::SniffForAbort().

Referenced by CopyToStream(), dng\_memory\_stream::CopyToStream(), Put\_real32(), Put\_real64(), Put\_uint16(), Put\_uint32(), Put\_uint64(), Put\_uint8(), PutZeros(), and dng\_iptc::Spool().

**6.56.3.25 void dng\_stream::Put\_int16 (int16 *x*) [inline]**

Put one 16-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

**Parameters:**

*x* One 16-bit integer.

References Put\_uint16().

**6.56.3.26 void dng\_stream::Put\_int32 (int32 *x*) [inline]**

Put one 32-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

**Parameters:**

*x* One 32-bit integer.

References Put\_uint32().

**6.56.3.27 void dng\_stream::Put\_int64 (int64 *x*) [inline]**

Put one 64-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

**Parameters:**

*x* One 64-bit integer.

References `Put_uint64()`.

**6.56.3.28** `void dng_stream::Put_int8 (int8 x) [inline]`

Put one 8-bit integer to stream and advance write position.

**Parameters:**

*x* One 8-bit integer.

References `Put_uint8()`.

**6.56.3.29** `void dng_stream::Put_real32 (real32 x)`

Put one 32-bit IEEE floating-point number to stream and advance write position. Byte swap if byte swapping is turned on.

**Parameters:**

*x* One 32-bit IEEE floating-point number.

References `Put()`, and `Put_uint32()`.

**6.56.3.30** `void dng_stream::Put_real64 (real64 x)`

Put one 64-bit IEEE floating-point number to stream and advance write position. Byte swap if byte swapping is turned on.

**Parameters:**

*x* One 64-bit IEEE floating-point number.

References `Put()`, and `Put_uint32()`.

**6.56.3.31** `void dng_stream::Put_uint16 (uint16 x)`

Put an unsigned 16-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

**Parameters:**

*x* One unsigned 16-bit integer.

References `Put()`.

Referenced by `Put_int16()`, `dng_iptc::Spool()`, `dng_image_writer::WriteDNG()`, and `dng_image_writer::WriteTIFFWithProfile()`.

**6.56.3.32 void `dng_stream::Put_uint32` (`uint32 x`)**

Put an unsigned 32-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

**Parameters:**

*x* One unsigned 32-bit integer.

References `Put()`.

Referenced by `Put_int32()`, `Put_real32()`, `Put_real64()`, `Put_uint64()`, `dng_image_writer::WriteDNG()`, and `dng_image_writer::WriteTIFFWithProfile()`.

**6.56.3.33 void `dng_stream::Put_uint64` (`uint64 x`)**

Put an unsigned 64-bit integer to stream and advance write position. Byte swap if byte swapping is turned on.

**Parameters:**

*x* One unsigned 64-bit integer.

References `Put()`, and `Put_uint32()`.

Referenced by `Put_int64()`.

**6.56.3.34 void `dng_stream::Put_uint8` (`uint8 x`) `[inline]`**

Put an unsigned 8-bit integer to stream and advance write position.

**Parameters:**

*x* One unsigned 8-bit integer.

References Length(), and Put().

Referenced by Put\_int8(), PutZeros(), and dng\_iptc::Spool().

**6.56.3.35 void dng\_stream::PutZeros (uint64 count)**

Writes the specified number of zero bytes to stream.

**Parameters:**

*count* Number of zero bytes to write.

References dng\_memory\_data::Buffer(), Put(), and Put\_uint8().

Referenced by PadAlign2(), and PadAlign4().

**6.56.3.36 void dng\_stream::SetBigEndian (bool bigEndian = true)**

Setter for whether data in stream is big endian.

**Parameters:**

*bigEndian* If true, data in stream is big endian.

Referenced by dng\_iptc::Parse(), dng\_info::Parse(), SetLittleEndian(), and dng\_iptc::Spool().

**6.56.3.37 void dng\_stream::SetLength (uint64 length)**

Set length of available data.

**Parameters:**

*length* Number of bytes of available data in stream.

References Flush(), and Length().

Referenced by DuplicateStream(), dng\_image\_writer::WriteDNG(), and dng\_image\_writer::WriteTIFFWithProfile().

**6.56.3.38** `void dng_stream::SetLittleEndian (bool littleEndian = true)`  
[inline]

Setter for whether data in stream is big endian.

**Parameters:**

*littleEndian* If true, data in stream is big endian.

References SetBigEndian().

Referenced by dng\_info::Parse().

**6.56.3.39** `void dng_stream::SetSniffer (dng_abort_sniffer * sniffer)`  
[inline]

Setter for sniffer associated with stream.

**Parameters:**

*sniffer* The new sniffer to use (or NULL for none).

**6.56.3.40** `void dng_stream::SetSwapBytes (bool swapBytes)` [inline]

Setter for whether stream is swapping byte order on input/output.

**Parameters:**

*swapBytes* If true, stream will swap byte order on input or output for future reads/writes.

**6.56.3.41** `void dng_stream::Skip (uint64 delta)` [inline]

Skip forward in stream.

**Parameters:**

*delta* Number of bytes to skip forward.

References Position(), and SetReadPosition().

**6.56.3.42** `dng_abort_sniffer* dng_stream::Sniffer () const` `[inline]`

Getter for sniffer associated with stream.

**Return values:**

*The* sniffer for this stream.

**6.56.3.43** `bool dng_stream::SwapBytes () const` `[inline]`

Getter for whether stream is swapping byte order on input/output.

**Return values:**

*If* true, data will be swapped on input/output.

**6.56.3.44** `int32 dng_stream::TagValue_int32 (uint32 tagType)`

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as a 32-bit integer.

**Parameters:**

*tagType* Tag type of data stored in stream.

**Return values:**

*One* 32-bit integer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get_int16()`, `Get_int32()`, `Get_int8()`, and `TagValue_real64()`.

Referenced by `TagValue_real64()`, and `TagValue_urational()`.

### 6.56.3.45 `real64 dng_stream::TagValue_real64 (uint32 tagType)`

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as a 64-bit IEEE floating-point number.

**Parameters:**

*tagType* Tag type of data stored in stream.

**Return values:**

*One* 64-bit IEEE floating-point number.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get_int32()`, `Get_real32()`, `Get_real64()`, `Get_uint32()`, `TagValue_int32()`, and `TagValue_uint32()`.

Referenced by `TagValue_int32()`, `TagValue_srational()`, `TagValue_uint32()`, and `TagValue_urational()`.

### 6.56.3.46 `dng_srational dng_stream::TagValue_srational (uint32 tagType)`

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as a `dng_srational`.

**Parameters:**

*tagType* Tag type of data stored in stream.

**Return values:**

*One* `dng_srational`.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get_int32()`, and `TagValue_real64()`.

### 6.56.3.47 uint32 dng\_stream::TagValue\_uint32 (uint32 tagType)

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as an unsigned 32-bit integer.

**Parameters:**

*tagType* Tag type of data stored in stream.

**Return values:**

*One* unsigned 32-bit integer.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get_uint16()`, `Get_uint32()`, `Get_uint8()`, and `TagValue_real64()`.

Referenced by `TagValue_real64()`, and `TagValue_urational()`.

### 6.56.3.48 dng\_urational dng\_stream::TagValue\_urational (uint32 tagType)

Get a value of size indicated by tag type from stream and advance read position. Byte swap if byte swapping is turned on and tag type is larger than a byte. Value is returned as a `dng_urational`.

**Parameters:**

*tagType* Tag type of data stored in stream.

**Return values:**

*One* `dng_urational`.

**Exceptions:**

*dng\_exception* with `fErrorCode` equal to `dng_error_end_of_file` if not enough data in stream.

References `Get_int32()`, `Get_uint32()`, `TagValue_int32()`, `TagValue_real64()`, and `TagValue_uint32()`.

The documentation for this class was generated from the following files:

- dng\_stream.h
- dng\_stream.cpp

## 6.57 dng\_tile\_buffer Class Reference

Class to get resource acquisition is instantiation behavior for tile buffers. Can be dirty or constant tile access.

```
#include <dng_image.h>
```

Inheritance diagram for dng\_tile\_buffer::

### Public Member Functions

- void **SetRefData** (void \*refData)
- void \* **GetRefData** () const

### Protected Member Functions

- [dng\\_tile\\_buffer](#) (const [dng\\_image](#) &image, const dng\_rect &tile, bool dirty)

### Protected Attributes

- const [dng\\_image](#) & **fImage**
- void \* **fRefData**

#### 6.57.1 Detailed Description

Class to get resource acquisition is instantiation behavior for tile buffers. Can be dirty or constant tile access.

#### 6.57.2 Constructor & Destructor Documentation

##### 6.57.2.1 dng\_tile\_buffer::dng\_tile\_buffer (const dng\_image & *image*, const dng\_rect & *tile*, bool *dirty*) [protected]

Obtain a tile from an image.

**Parameters:**

*image* Image tile will come from.

*tile* Rectangle denoting extent of tile.

*dirty* Flag indicating whether this is read-only or read-write access.

The documentation for this class was generated from the following files:

- [dng\\_image.h](#)
- `dng_image.cpp`

## 6.58 `dng_time_zone` Class Reference

Class for holding a time zone.

```
#include <dng_date_time.h>
```

**Public Member Functions**

- void **Clear** ()
- void **SetOffsetHours** (int32 offset)
- void **SetOffsetMinutes** (int32 offset)
- void **SetOffsetSeconds** (int32 offset)
- bool **IsValid** () const
- bool **NotValid** () const
- int32 **OffsetMinutes** () const
- bool **IsExactHourOffset** () const
- int32 **ExactHourOffset** () const
- dng\_string **Encode\_ISO\_8601** () const

### 6.58.1 Detailed Description

Class for holding a time zone.

The documentation for this class was generated from the following files:

- [dng\\_date\\_time.h](#)
- `dng_date_time.cpp`

## 6.59 `dng_tone_curve_acr3_default` Class Reference

Default ACR3 tone curve.

```
#include <dng_render.h>
```

Inheritance diagram for `dng_tone_curve_acr3_default`:

### Public Member Functions

- virtual `real64 Evaluate` (`real64 x`) const  
*Returns output value for a given input tone.*
- virtual `real64 EvaluateInverse` (`real64 x`) const  
*Returns nearest input value for a given output tone.*

### Static Public Member Functions

- static const `dng_1d_function & Get` ()

#### 6.59.1 Detailed Description

Default ACR3 tone curve.

The documentation for this class was generated from the following files:

- [dng\\_render.h](#)
- [dng\\_render.cpp](#)

## 6.60 `dng_vignette_radial_params` Class Reference

Radially-symmetric vignette (peripheral illuminational falloff) correction parameters.

```
#include <dng_lens_correction.h>
```

### Public Member Functions

- `dng_vignette_radial_params` (`const std::vector< real64 > &params`, `const dng_point_real64 &center`)
- `bool IsNOP` () const
- `bool IsValid` () const
- `void Dump` () const

### Public Attributes

- `std::vector< real64 > fParams`
- `dng_point_real64 fCenter`

### Static Public Attributes

- `static const uint32 kNumTerms = 5`

#### 6.60.1 Detailed Description

Radially-symmetric vignette (peripheral illuminational falloff) correction parameters.

The documentation for this class was generated from the following files:

- `dng_lens_correction.h`
- `dng_lens_correction.cpp`

## 6.61 dng\_warp\_params Class Reference

Abstract base class holding common warp opcode parameters (e.g., number of planes, optical center) and common warp routines.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for `dng_warp_params`:

### Public Member Functions

- `dng_warp_params` (uint32 planes, const `dng_point_real64` &fCenter)
- virtual bool **IsNOPAll** () const
- virtual bool **IsNOP** (uint32 plane) const
- virtual bool **IsRadNOPAll** () const
- virtual bool **IsRadNOP** (uint32 plane) const
- virtual bool **IsTanNOPAll** () const
- virtual bool **IsTanNOP** (uint32 plane) const
- virtual bool **IsValid** () const
- virtual bool **IsValidForNegative** (const `dng_negative` &negative) const
- virtual void **PropagateToAllPlanes** (uint32 totalPlanes)=0
- virtual `real64` **Evaluate** (uint32 plane, `real64` r) const =0
- virtual `real64` **EvaluateInverse** (uint32 plane, `real64` r) const

- virtual real64 **EvaluateRatio** (uint32 plane, real64 r2) const =0
- virtual dng\_point\_real64 **EvaluateTangential** (uint32 plane, real64 r2, const dng\_point\_real64 &diff, const dng\_point\_real64 &diff2) const =0
- dng\_point\_real64 **EvaluateTangential2** (uint32 plane, const dng\_point\_real64 &diff) const
- dng\_point\_real64 **EvaluateTangential3** (uint32 plane, real64 r2, const dng\_point\_real64 &diff) const
- virtual real64 **MaxSrcRadiusGap** (real64 maxDstGap) const =0
- virtual dng\_point\_real64 **MaxSrcTanGap** (dng\_point\_real64 minDst, dng\_point\_real64 maxDst) const =0
- virtual void **Dump** () const

### Public Attributes

- uint32 **fPlanes**
- dng\_point\_real64 **fCenter**

#### 6.61.1 Detailed Description

Abstract base class holding common warp opcode parameters (e.g., number of planes, optical center) and common warp routines.

The documentation for this class was generated from the following files:

- dng\_lens\_correction.h
- dng\_lens\_correction.cpp

## 6.62 dng\_warp\_params\_fisheye Class Reference

Warp parameters for fisheye camera model (radial component only). Note the restrictions described below.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for dng\_warp\_params\_fisheye::

### Public Member Functions

- **dng\_warp\_params\_fisheye** (uint32 planes, const dng\_vector radParams[], const dng\_point\_real64 &fCenter)
- virtual bool **IsRadNOP** (uint32 plane) const
- virtual bool **IsTanNOP** (uint32 plane) const

- virtual bool **IsValid** () const
- virtual void **PropagateToAllPlanes** (uint32 totalPlanes)
- virtual real64 **Evaluate** (uint32 plane, real64 r) const
- virtual real64 **EvaluateRatio** (uint32 plane, real64 r2) const
- virtual dng\_point\_real64 **EvaluateTangential** (uint32 plane, real64 r2, const dng\_point\_real64 &diff, const dng\_point\_real64 &diff2) const
- virtual real64 **MaxSrcRadiusGap** (real64 maxDstGap) const
- virtual dng\_point\_real64 **MaxSrcTanGap** (dng\_point\_real64 minDst, dng\_point\_real64 maxDst) const
- virtual void **Dump** () const

### Public Attributes

- dng\_vector **fRadParams** [[kMaxColorPlanes](#)]

#### 6.62.1 Detailed Description

Warp parameters for fisheye camera model (radial component only). Note the restrictions described below.

The documentation for this class was generated from the following files:

- dng\_lens\_correction.h
- dng\_lens\_correction.cpp

### 6.63 dng\_warp\_params\_rectilinear Class Reference

Warp parameters for pinhole perspective rectilinear (not fisheye) camera model. Supports radial and tangential (decentering) distortion correction parameters.

```
#include <dng_lens_correction.h>
```

Inheritance diagram for dng\_warp\_params\_rectilinear::

### Public Member Functions

- **dng\_warp\_params\_rectilinear** (uint32 planes, const dng\_vector radParams[], const dng\_vector tanParams[], const dng\_point\_real64 &fCenter)
- virtual bool **IsRadNOP** (uint32 plane) const
- virtual bool **IsTanNOP** (uint32 plane) const
- virtual bool **IsValid** () const

- virtual void **PropagateToAllPlanes** (uint32 totalPlanes)
- virtual real64 **Evaluate** (uint32 plane, real64 r) const
- virtual real64 **EvaluateRatio** (uint32 plane, real64 r2) const
- virtual dng\_point\_real64 **EvaluateTangential** (uint32 plane, real64 r2, const dng\_point\_real64 &diff, const dng\_point\_real64 &diff2) const
- virtual real64 **MaxSrcRadiusGap** (real64 maxDstGap) const
- virtual dng\_point\_real64 **MaxSrcTanGap** (dng\_point\_real64 minDst, dng\_point\_real64 maxDst) const
- virtual void **Dump** () const

### Public Attributes

- dng\_vector **fRadParams** [[kMaxColorPlanes](#)]
- dng\_vector **fTanParams** [[kMaxColorPlanes](#)]

#### 6.63.1 Detailed Description

Warp parameters for pinhole perspective rectilinear (not fisheye) camera model. Supports radial and tangential (decentering) distortion correction parameters.

Note the restrictions described below.

The documentation for this class was generated from the following files:

- dng\_lens\_correction.h
- dng\_lens\_correction.cpp

## 7 File Documentation

### 7.1 dng\_1d\_function.h File Reference

#### Classes

- class [dng\\_1d\\_function](#)  
*A 1D floating-point function.*
- class [dng\\_1d\\_identity](#)  
*An identity ( $x \rightarrow y$  such that  $x == y$  for all  $x$ ) mapping function.*
- class [dng\\_1d\\_concatenate](#)  
*A [dng\\_1d\\_function](#) that represents the composition (curry) of two other [dng\\_1d\\_functions](#).*

- class [dng\\_1d\\_inverse](#)  
*A [dng\\_1d\\_function](#) that represents the inverse of another [dng\\_1d\\_function](#).*

### 7.1.1 Detailed Description

Classes for a 1D floating-point to floating-point function abstraction.

## 7.2 dng\_1d\_table.h File Reference

### Classes

- class [dng\\_1d\\_table](#)  
*A 1D floating-point lookup table using linear interpolation.*

### 7.2.1 Detailed Description

Definition of a lookup table based 1D floating-point to floating-point function abstraction using linear interpolation.

## 7.3 dng\_abort\_sniffer.h File Reference

### Classes

- class **dng\_set\_minimum\_priority**
- class [dng\\_abort\\_sniffer](#)  
*Class for signaling user cancellation and receiving progress updates.*
- class [dng\\_sniffer\\_task](#)  
*Class to establish scope of a named subtask in DNG processing.*

### Enumerations

- enum **dng\_priority** {  
    **dng\_priority\_low**,   **dng\_priority\_medium**,   **dng\_priority\_high**,   **dng\_**  
    **priority\_count**,

```
dng_priority_minimum = dng_priority_low, dng_priority_maximum = dng_
priority_high }
```

### 7.3.1 Detailed Description

Classes supporting user cancellation and progress tracking.

## 7.4 dng\_area\_task.h File Reference

### Classes

- class [dng\\_area\\_task](#)

*Abstract class for rectangular processing operations with support for partitioning across multiple processing resources and observing memory constraints.*

### 7.4.1 Detailed Description

Class to handle partitioning a rectangular image processing operation taking into account multiple processing resources and memory constraints.

## 7.5 dng\_assertions.h File Reference

### Defines

- #define [DNG\\_ASSERT\(x, y\)](#)
- #define [DNG\\_REQUIRE\(condition, msg\)](#)
- #define [DNG\\_REPORT\(x\) DNG\\_ASSERT \(false, x\)](#)

### 7.5.1 Detailed Description

Conditionally compiled assertion check support.

### 7.5.2 Define Documentation

#### 7.5.2.1 #define DNG\_ASSERT(x, y)

Conditionally compiled macro to check an assertion and display a message if it fails and assertions are compiled in via qDNGDebug

**Parameters:**

- x* Predicate which must be true.
- y* String to display if *x* is not true.

Referenced by `dng_negative::AnalogBalance()`, `dng_color_spec::CameraToPCS()`, `dng_color_spec::CameraWhite()`, `dng_pixel_buffer::DirtyPixel()`, `dng_1d_table::Interpolate()`, `dng_pixel_buffer::SetConstant_int16()`, `dng_pixel_buffer::SetConstant_real32()`, `dng_pixel_buffer::SetConstant_uint16()`, `dng_pixel_buffer::SetConstant_uint32()`, `dng_pixel_buffer::SetConstant_uint8()`, `dng_ipcc::Spool()`, and `dng_color_spec::WhiteXY()`.

### 7.5.2.2 #define DNG\_REPORT(*x*) DNG\_ASSERT (*false*, *x*)

Macro to display an informational message

**Parameters:**

- x* String to display.

### 7.5.2.3 #define DNG\_REQUIRE(*condition*, *msg*)

**Value:**

```
do
    {
        if (!(condition))
        {
            ThrowProgramError (msg);
        }
    }
while (0)
```

Conditionally compiled macro to check an assertion, display a message, and throw an exception if it fails and assertions are compiled in via qDNGDebug

**Parameters:**

- condition* Predicate which must be true.
- msg* String to display if condition is not true.

## 7.6 dng\_auto\_ptr.h File Reference

**Classes**

- class [AutoPtr< T >](#)

*A class intended to be used in stack scope to hold a pointer from new. The held pointer will be deleted automatically if the scope is left without calling Release on the [AutoPtr](#) first.*

### 7.6.1 Detailed Description

Class to implement std::auto\_ptr like functionality even on platforms which do not have a full Standard C++ library.

## 7.7 dng\_bottlenecks.h File Reference

**Classes**

- struct **dng\_suite**

**Typedefs**

- typedef void( **ZeroBytesProc** )(void \*dPtr, uint32 count)
- typedef void( **CopyBytesProc** )(const void \*sPtr, void \*dPtr, uint32 count)
- typedef void( **SwapBytes16Proc** )(uint16 \*dPtr, uint32 count)
- typedef void( **SwapBytes32Proc** )(uint32 \*dPtr, uint32 count)
- typedef void( **SetArea8Proc** )(uint8 \*dPtr, uint8 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- typedef void( **SetArea16Proc** )(uint16 \*dPtr, uint16 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- typedef void( **SetArea32Proc** )(uint32 \*dPtr, uint32 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- typedef void( **CopyArea8Proc** )(const uint8 \*sPtr, uint8 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)

- typedef void( **CopyArea16Proc** )(const uint16 \*sPtr, uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void( **CopyArea32Proc** )(const uint32 \*sPtr, uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void( **CopyArea8\_16Proc** )(const uint8 \*sPtr, uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void( **CopyArea8\_S16Proc** )(const uint8 \*sPtr, int16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void( **CopyArea8\_32Proc** )(const uint8 \*sPtr, uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void( **CopyArea16\_S16Proc** )(const uint16 \*sPtr, int16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void( **CopyArea16\_32Proc** )(const uint16 \*sPtr, uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void( **CopyArea8\_R32Proc** )(const uint8 \*sPtr, real32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixel-Range)
- typedef void( **CopyArea16\_R32Proc** )(const uint16 \*sPtr, real32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixel-Range)
- typedef void( **CopyAreaS16\_R32Proc** )(const int16 \*sPtr, real32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixel-Range)
- typedef void( **CopyAreaR32\_8Proc** )(const real32 \*sPtr, uint8 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixel-Range)
- typedef void( **CopyAreaR32\_16Proc** )(const real32 \*sPtr, uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixel-Range)
- typedef void( **CopyAreaR32\_S16Proc** )(const real32 \*sPtr, int16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixel-Range)

- typedef void( **RepeatArea8Proc** )(const uint8 \*sPtr, uint8 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- typedef void( **RepeatArea16Proc** )(const uint16 \*sPtr, uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- typedef void( **RepeatArea32Proc** )(const uint32 \*sPtr, uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- typedef void( **ShiftRight16Proc** )(uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 shift)
- typedef void( **BilinearRow16Proc** )(const uint16 \*sPtr, uint16 \*dPtr, uint32 cols, uint32 patPhase, uint32 patCount, const uint32 \*kernCounts, const int32 \*const \*kernOffsets, const uint16 \*const \*kernWeights, uint32 sShift)
- typedef void( **BilinearRow32Proc** )(const real32 \*sPtr, real32 \*dPtr, uint32 cols, uint32 patPhase, uint32 patCount, const uint32 \*kernCounts, const int32 \*const \*kernOffsets, const real32 \*const \*kernWeights, uint32 sShift)
- typedef void( **BaselineABCtoRGBProc** )(const real32 \*sPtrA, const real32 \*sPtrB, const real32 \*sPtrC, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const dng\_vector &cameraWhite, const dng\_matrix &cameraToRGB)
- typedef void( **BaselineABCDtoRGBProc** )(const real32 \*sPtrA, const real32 \*sPtrB, const real32 \*sPtrC, const real32 \*sPtrD, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const dng\_vector &cameraWhite, const dng\_matrix &cameraToRGB)
- typedef void( **BaselineHueSatMapProc** )(const real32 \*sPtrR, const real32 \*sPtrG, const real32 \*sPtrB, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const dng\_hue\_sat\_map &lut)
- typedef void( **BaselineGrayToRGBProc** )(const real32 \*sPtrR, const real32 \*sPtrG, const real32 \*sPtrB, real32 \*dPtrG, uint32 count, const dng\_matrix &matrix)
- typedef void( **BaselineRGBtoRGBProc** )(const real32 \*sPtrR, const real32 \*sPtrG, const real32 \*sPtrB, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const dng\_matrix &matrix)
- typedef void( **Baseline1DTableProc** )(const real32 \*sPtr, real32 \*dPtr, uint32 count, const [dng\\_1d\\_table](#) &table)
- typedef void( **BaselineRGBToneProc** )(const real32 \*sPtrR, const real32 \*sPtrG, const real32 \*sPtrB, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const [dng\\_1d\\_table](#) &table)
- typedef void( **ResampleDown16Proc** )(const uint16 \*sPtr, uint16 \*dPtr, uint32 sCount, int32 sRowStep, const int16 \*wPtr, uint32 wCount, uint32 pixelRange)
- typedef void( **ResampleDown32Proc** )(const real32 \*sPtr, real32 \*dPtr, uint32 sCount, int32 sRowStep, const real32 \*wPtr, uint32 wCount)
- typedef void( **ResampleAcross16Proc** )(const uint16 \*sPtr, uint16 \*dPtr, uint32 dCount, const int32 \*coord, const int16 \*wPtr, uint32 wCount, uint32 wStep, uint32 pixelRange)

- typedef void( **ResampleAcross32Proc** )(const real32 \*sPtr, real32 \*dPtr, uint32 dCount, const int32 \*coord, const real32 \*wPtr, uint32 wCount, uint32 wStep)
- typedef bool( **EqualBytesProc** )(const void \*sPtr, const void \*dPtr, uint32 count)
- typedef bool( **EqualArea8Proc** )(const uint8 \*sPtr, const uint8 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef bool( **EqualArea16Proc** )(const uint16 \*sPtr, const uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef bool( **EqualArea32Proc** )(const uint32 \*sPtr, const uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- typedef void( **VignetteMask16Proc** )(uint16 \*mPtr, uint32 rows, uint32 cols, int32 rowStep, int64 offsetH, int64 offsetV, int64 stepH, int64 stepV, uint32 tBits, const uint16 \*table)
- typedef void( **Vignette16Proc** )(int16 \*sPtr, const uint16 \*mPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sPlaneStep, int32 mRowStep, uint32 mBits)
- typedef void( **MapArea16Proc** )(uint16 \*dPtr, uint32 count0, uint32 count1, uint32 count2, int32 step0, int32 step1, int32 step2, const uint16 \*map)

## Functions

- void **DoZeroBytes** (void \*dPtr, uint32 count)
- void **DoCopyBytes** (const void \*sPtr, void \*dPtr, uint32 count)
- void **DoSwapBytes16** (uint16 \*dPtr, uint32 count)
- void **DoSwapBytes32** (uint32 \*dPtr, uint32 count)
- void **DoSetArea8** (uint8 \*dPtr, uint8 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- void **DoSetArea16** (uint16 \*dPtr, uint16 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- void **DoSetArea32** (uint32 \*dPtr, uint32 value, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep)
- void **DoCopyArea8** (const uint8 \*sPtr, uint8 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea16** (const uint16 \*sPtr, uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea32** (const uint32 \*sPtr, uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)

- void **DoCopyArea8\_16** (const uint8 \*sPtr, uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea8\_S16** (const uint8 \*sPtr, int16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea8\_32** (const uint8 \*sPtr, uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea16\_S16** (const uint16 \*sPtr, int16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea16\_32** (const uint16 \*sPtr, uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- void **DoCopyArea8\_R32** (const uint8 \*sPtr, real32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyArea16\_R32** (const uint16 \*sPtr, real32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyAreaS16\_R32** (const int16 \*sPtr, real32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyAreaR32\_8** (const real32 \*sPtr, uint8 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyAreaR32\_16** (const real32 \*sPtr, uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoCopyAreaR32\_S16** (const real32 \*sPtr, int16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep, uint32 pixelRange)
- void **DoRepeatArea8** (const uint8 \*sPtr, uint8 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- void **DoRepeatArea16** (const uint16 \*sPtr, uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- void **DoRepeatArea32** (const uint32 \*sPtr, uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 repeatV, uint32 repeatH, uint32 phaseV, uint32 phaseH)
- void **DoShiftRight16** (uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 rowStep, int32 colStep, int32 planeStep, uint32 shift)

- void **DoBilinearRow16** (const uint16 \*sPtr, uint16 \*dPtr, uint32 cols, uint32 patPhase, uint32 patCount, const uint32 \*kernCounts, const int32 \*const \*kernOffsets, const uint16 \*const \*kernWeights, uint32 sShift)
- void **DoBilinearRow32** (const real32 \*sPtr, real32 \*dPtr, uint32 cols, uint32 patPhase, uint32 patCount, const uint32 \*kernCounts, const int32 \*const \*kernOffsets, const real32 \*const \*kernWeights, uint32 sShift)
- void **DoBaselineABCtoRGB** (const real32 \*sPtrA, const real32 \*sPtrB, const real32 \*sPtrC, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const dng\_vector &cameraWhite, const dng\_matrix &cameraToRGB)
- void **DoBaselineABCDtoRGB** (const real32 \*sPtrA, const real32 \*sPtrB, const real32 \*sPtrC, const real32 \*sPtrD, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const dng\_vector &cameraWhite, const dng\_matrix &cameraToRGB)
- void **DoBaselineHueSatMap** (const real32 \*sPtrR, const real32 \*sPtrG, const real32 \*sPtrB, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const dng\_hue\_sat\_map &lut)
- void **DoBaselineRGBtoGray** (const real32 \*sPtrR, const real32 \*sPtrG, const real32 \*sPtrB, real32 \*dPtrG, uint32 count, const dng\_matrix &matrix)
- void **DoBaselineRGBtoRGB** (const real32 \*sPtrR, const real32 \*sPtrG, const real32 \*sPtrB, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const dng\_matrix &matrix)
- void **DoBaseline1DTable** (const real32 \*sPtr, real32 \*dPtr, uint32 count, const [dng\\_1d\\_table](#) &table)
- void **DoBaselineRGBTone** (const real32 \*sPtrR, const real32 \*sPtrG, const real32 \*sPtrB, real32 \*dPtrR, real32 \*dPtrG, real32 \*dPtrB, uint32 count, const [dng\\_1d\\_table](#) &table)
- void **DoResampleDown16** (const uint16 \*sPtr, uint16 \*dPtr, uint32 sCount, int32 sRowStep, const int16 \*wPtr, uint32 wCount, uint32 pixelRange)
- void **DoResampleDown32** (const real32 \*sPtr, real32 \*dPtr, uint32 sCount, int32 sRowStep, const real32 \*wPtr, uint32 wCount)
- void **DoResampleAcross16** (const uint16 \*sPtr, uint16 \*dPtr, uint32 dCount, const int32 \*coord, const int16 \*wPtr, uint32 wCount, uint32 wStep, uint32 pixelRange)
- void **DoResampleAcross32** (const real32 \*sPtr, real32 \*dPtr, uint32 dCount, const int32 \*coord, const real32 \*wPtr, uint32 wCount, uint32 wStep)
- bool **DoEqualBytes** (const void \*sPtr, const void \*dPtr, uint32 count)
- bool **DoEqualArea8** (const uint8 \*sPtr, const uint8 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- bool **DoEqualArea16** (const uint16 \*sPtr, const uint16 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)
- bool **DoEqualArea32** (const uint32 \*sPtr, const uint32 \*dPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sColStep, int32 sPlaneStep, int32 dRowStep, int32 dColStep, int32 dPlaneStep)

- void **DoVignetteMask16** (uint16 \*mPtr, uint32 rows, uint32 cols, int32 rowStep, int64 offsetH, int64 offsetV, int64 stepH, int64 stepV, uint32 tBits, const uint16 \*table)
- void **DoVignette16** (int16 \*sPtr, const uint16 \*mPtr, uint32 rows, uint32 cols, uint32 planes, int32 sRowStep, int32 sPlaneStep, int32 mRowStep, uint32 mBits)
- void **DoMapArea16** (uint16 \*dPtr, uint32 count0, uint32 count1, uint32 count2, int32 step0, int32 step1, int32 step2, const uint16 \*map)

### Variables

- dng\_suite **gDNGSuite**

### 7.7.1 Detailed Description

Indirection mechanism for performance-critical routines that might be replaced with hand-optimized or hardware-specific implementations.

## 7.8 dng\_camera\_profile.h File Reference

### Classes

- class **dng\_camera\_profile\_id**
- class [dng\\_camera\\_profile](#)  
*Container for DNG camera color profile and calibration data.*

### Functions

- void **SplitCameraProfileName** (const dng\_string &name, dng\_string &baseName, int32 &version)

### Variables

- const char \* **kProfileName\_Embedded**
- const char \* **kAdobeCalibrationSignature**

### 7.8.1 Detailed Description

Support for DNG camera color profile information. Per the [DNG 1.1.0 specification](#), a DNG file can store up to two sets of color profile information for a camera in the DNG file from that camera. The second set is optional and when there are two sets, they represent profiles made under different illumination.

Profiling information is optionally separated into two parts. One part represents a profile for a reference camera. (`ColorMatrix1` and `ColorMatrix2` here.) The second is a per-camera calibration that takes into account unit-to-unit variation. This is designed to allow replacing the reference color matrix with one of one's own construction while maintaining any unit-specific calibration the camera manufacturer may have provided.

See Appendix 6 of the [DNG 1.1.0 specification](#) for more information.

## 7.9 `dng_color_space.h` File Reference

### Classes

- class [`dng\_function\_GammaEncode\_sRGB`](#)  
*A `dng_Id_function` for gamma encoding in sRGB color space.*
- class [`dng\_function\_GammaEncode\_1\_8`](#)  
*A `dng_Id_function` for gamma encoding with 1.8 gamma.*
- class [`dng\_function\_GammaEncode\_2\_2`](#)  
*A `dng_Id_function` for gamma encoding with 2.2 gamma.*
- class [`dng\_color\_space`](#)  
*An abstract color space.*
- class [`dng\_space\_sRGB`](#)  
*Singleton class for sRGB color space.*
- class [`dng\_space\_AdobeRGB`](#)  
*Singleton class for AdobeRGB color space.*
- class [`dng\_space\_ColorMatch`](#)  
*Singleton class for ColorMatch color space.*
- class [`dng\_space\_ProPhoto`](#)  
*Singleton class for ProPhoto RGB color space.*
- class [`dng\_space\_GrayGamma18`](#)

*Singleton class for gamma 1.8 grayscale color space.*

- class [dng\\_space\\_GrayGamma22](#)  
*Singleton class for gamma 2.2 grayscale color space.*
- class **dng\_space\_fakeRGB**

### 7.9.1 Detailed Description

Standard gamma functions and color spaces used within the DNG SDK.

## 7.10 `dng_color_spec.h` File Reference

### Classes

- class [dng\\_color\\_spec](#)

### Functions

- `dng_matrix_3by3` [MapWhiteMatrix](#) (`const dng_xy_coord &white1`, `const dng_xy_coord &white2`)  
*Compute a 3x3 matrix which maps colors from white point `white1` to white point `white2`.*

### 7.10.1 Detailed Description

Class for holding a specific color transform.

### 7.10.2 Function Documentation

#### 7.10.2.1 `dng_matrix_3by3` [MapWhiteMatrix](#) (`const dng_xy_coord &white1`, `const dng_xy_coord &white2`)

Compute a 3x3 matrix which maps colors from white point `white1` to white point `white2`.

Uses linearized Bradford adaptation matrix to compute a mapping from colors measured with one white point (`white1`) to another (`white2`).

## 7.11 dng\_date\_time.h File Reference

### Classes

- class [dng\\_date\\_time](#)  
*Class for holding a date/time and converting to and from relevant date/time formats.*
- class [dng\\_time\\_zone](#)  
*Class for holding a time zone.*
- class [dng\\_date\\_time\\_info](#)  
*Class for holding complete data/time/zone information.*
- class [dng\\_date\\_time\\_storage\\_info](#)  
*Store file offset from which date was read.*

### Enumerations

- enum [dng\\_date\\_time\\_format](#) { [dng\\_date\\_time\\_format\\_unknown](#) = 0, [dng\\_date\\_time\\_format\\_exif](#) = 1, [dng\\_date\\_time\\_format\\_unix\\_little\\_endian](#) = 2, [dng\\_date\\_time\\_format\\_unix\\_big\\_endian](#) = 3 }
- Tag to encode date representation format.*

### Functions

- void [CurrentDateTimeAndZone](#) ([dng\\_date\\_time\\_info](#) &info)
- void [DecodeUnixTime](#) (uint32 unixTime, [dng\\_date\\_time](#) &dt)  
*Convert UNIX "seconds since Jan 1, 1970" time to a [dng\\_date\\_time](#).*
- [dng\\_time\\_zone LocalTimeZone](#) (const [dng\\_date\\_time](#) &dt)

#### 7.11.1 Detailed Description

Functions and classes for working with dates and times in DNG files.

## 7.11.2 Enumeration Type Documentation

### 7.11.2.1 enum dng\_date\_time\_format

Tag to encode date representation format.

#### Enumerator:

- dng\_date\_time\_format\_exif* Date format not known.
- dng\_date\_time\_format\_unix\_little\_endian* EXIF date string.
- dng\_date\_time\_format\_unix\_big\_endian* 32-bit UNIX time as 4-byte little endian

## 7.11.3 Function Documentation

### 7.11.3.1 void CurrentDateTimeAndZone (dng\_date\_time\_info & info)

Get the current date/time and timezone.

#### Parameters:

- info* Receives current data/time/zone.

### 7.11.3.2 dng\_time\_zone LocalTimeZone (const dng\_date\_time & dt)

Return timezone of current location at a given date.

#### Parameters:

- dt* Date at which to compute timezone difference. (For example, used to determine Daylight Savings, etc.)

#### Return values:

- Time* zone for date/time dt.

References `dng_date_time::IsValid()`.

## 7.12 `dng_errors.h` File Reference

### Typedefs

- typedef int32 [dng\\_error\\_code](#)  
*Type for all errors used in DNG SDK. Generally held inside a [dng\\_exception](#).*

### Enumerations

- enum {  
`dng_error_none = 0, dng_error_unknown = 100000, dng_error_not_yet_ - implemented, dng_error_silent,`  
`dng_error_user_canceled, dng_error_host_insufficient, dng_error_ - memory, dng_error_bad_format,`  
`dng_error_matrix_math, dng_error_open_file, dng_error_read_file, dng_ - error_write_file,`  
`dng_error_end_of_file, dng_error_file_is_damaged, dng_error_image_ - too_big_dng, dng_error_image_too_big_tiff }`

#### 7.12.1 Detailed Description

Error code values.

## 7.13 `dng_exceptions.h` File Reference

### Classes

- class [dng\\_exception](#)  
*All exceptions thrown by the DNG SDK use this exception class.*

### Functions

- void [ReportWarning](#) (const char \*message, const char \*sub\_message=NULL)  
*Display a warning message. Note that this may just eat the message.*
- void [ReportError](#) (const char \*message, const char \*sub\_message=NULL)  
*Display an error message. Note that this may just eat the message.*

- void `Throw_dng_error` (`dng_error_code` err, const char \*message=NULL, const char \*sub\_message=NULL, bool silent=false)  
*Throw an exception based on an arbitrary error code.*
- void `Fail_dng_error` (`dng_error_code` err)  
*Convenience function to throw `dng_exception` with error code if error\_code is not `dng_error_none`.*
- void `ThrowProgramError` (const char \*sub\_message=NULL)  
*Convenience function to throw `dng_exception` with error code `dng_error_unknown`.*
- void `ThrowNotYetImplemented` (const char \*sub\_message=NULL)  
*Convenience function to throw `dng_exception` with error code `dng_error_not_yet_implemented`.*
- void `ThrowSilentError` ()  
*Convenience function to throw `dng_exception` with error code `dng_error_silent`.*
- void `ThrowUserCanceled` ()  
*Convenience function to throw `dng_exception` with error code `dng_error_user_canceled`.*
- void `ThrowHostInsufficient` (const char \*sub\_message=NULL)  
*Convenience function to throw `dng_exception` with error code `dng_error_host_insufficient`.*
- void `ThrowMemoryFull` (const char \*sub\_message=NULL)  
*Convenience function to throw `dng_exception` with error code `dng_error_memory`.*
- void `ThrowBadFormat` (const char \*sub\_message=NULL)  
*Convenience function to throw `dng_exception` with error code `dng_error_bad_format`.*
- void `ThrowMatrixMath` (const char \*sub\_message=NULL)  
*Convenience function to throw `dng_exception` with error code `dng_error_matrix_math`.*
- void `ThrowOpenFile` (const char \*sub\_message=NULL, bool silent=false)  
*Convenience function to throw `dng_exception` with error code `dng_error_open_file`.*
- void `ThrowReadFile` (const char \*sub\_message=NULL)  
*Convenience function to throw `dng_exception` with error code `dng_error_read_file`.*

- void [ThrowWriteFile](#) (const char \*sub\_message=NULL)  
*Convenience function to throw [dng\\_exception](#) with error code `dng_error_write_file`.*
- void [ThrowEndOfFile](#) (const char \*sub\_message=NULL)  
*Convenience function to throw [dng\\_exception](#) with error code `dng_error_end_of_file`.*
- void [ThrowFileIsDamaged](#) ()  
*Convenience function to throw [dng\\_exception](#) with error code `dng_error_file_is_damaged`.*
- void [ThrowImageTooBigDNG](#) ()  
*Convenience function to throw [dng\\_exception](#) with error code `dng_error_image_too_big_dng`.*
- void [ThrowImageTooBigTIFF](#) ()  
*Convenience function to throw [dng\\_exception](#) with error code `dng_error_image_too_big_tiff`.*

### 7.13.1 Detailed Description

C++ exception support for DNG SDK.

## 7.14 dng\_exif.h File Reference

### Classes

- class [dng\\_exif](#)  
*Container class for parsing and holding EXIF tags.*

### 7.14.1 Detailed Description

EXIF read access support. See the [EXIF specification](#) for full description of tags.

## 7.15 `dng_fast_module.h` File Reference

### 7.15.1 Detailed Description

Include file to set optimization to highest level for performance-critical routines. Normal files should have optimization set to normal level to save code size as there is less cache pollution this way.

## 7.16 `dng_file_stream.h` File Reference

### Classes

- class [dng\\_file\\_stream](#)  
*A stream to/from a disk file. See [dng\\_stream](#) for read/write interface.*

### 7.16.1 Detailed Description

Simple, portable, file read/write support.

## 7.17 `dng_filter_task.h` File Reference

### Classes

- class [dng\\_filter\\_task](#)  
*Represents a task which filters an area of a source [dng\\_image](#) to an area of a destination [dng\\_image](#).*

### 7.17.1 Detailed Description

Specialization of [dng\\_area\\_task](#) for processing an area from one [dng\\_image](#) to an area of another.

## 7.18 `dng_fingerprint.h` File Reference

### Classes

- class [dng\\_fingerprint](#)  
*Container fingerprint (MD5 only at present).*

- class `dng_md5_printer`
- class `dng_md5_printer_stream`

### 7.18.1 Detailed Description

Fingerprint (cryptographic hashing) support for generating strong hashes of image data.

## 7.19 dng\_flags.h File Reference

### Defines

- #define `qDNGDebug` 0
- #define `qDNGLittleEndian` !qDNGBigEndian
- #define `qDNG64Bit` 0
- #define `qDNGThreadSafe` (qMacOS || qWinOS)
- #define `qDNGValidateTarget` 0
- #define `qDNGValidate` qDNGValidateTarget
- #define `qDNGPrintMessages` qDNGValidate

### 7.19.1 Detailed Description

Conditional compilation flags for DNG SDK.

All conditional compilation macros for the DNG SDK begin with a lowercase 'q'.

## 7.20 dng\_globals.h File Reference

### 7.20.1 Detailed Description

Definitions of global variables controlling DNG SDK behavior. Currently only used for validation control.

## 7.21 dng\_host.h File Reference

### Classes

- class `dng_host`

*The main class for communication between the application and the DNG SDK. Used to customize memory allocation and other behaviors.*

### 7.21.1 Detailed Description

Class definition for [dng\\_host](#), initial point of contact and control between host application and DNG SDK.

## 7.22 **dng\_ifd.h File Reference**

### Classes

- class [dng\\_preview\\_info](#)
- class [dng\\_ifd](#)

*Container for a single image file directory of a digital negative.*

### 7.22.1 Detailed Description

DNG image file directory support.

## 7.23 **dng\_image.h File Reference**

### Classes

- class [dng\\_tile\\_buffer](#)  
*Class to get resource acquisition is instantiation behavior for tile buffers. Can be dirty or constant tile access.*
- class [dng\\_const\\_tile\\_buffer](#)  
*Class to get resource acquisition is instantiation behavior for constant (read-only) tile buffers.*
- class [dng\\_dirty\\_tile\\_buffer](#)  
*Class to get resource acquisition is instantiation behavior for dirty (writable) tile buffers.*
- class [dng\\_image](#)  
*Base class for holding image data in DNG SDK. See [dng\\_simple\\_image](#) for derived class most often used in DNG SDK.*

### 7.23.1 Detailed Description

Support for working with image data in DNG SDK.

## 7.24 `dng_image_writer.h` File Reference

### Classes

- class `dng_resolution`
- class `tiff_tag`
- class `tag_data_ptr`
- class `tag_string`
- class `tag_encoded_text`
- class `tag_uint8`
- class `tag_uint8_ptr`
- class `tag_uint16`
- class `tag_int16_ptr`
- class `tag_uint16_ptr`
- class `tag_uint32`
- class `tag_uint32_ptr`
- class `tag_urational`
- class `tag_urational_ptr`
- class `tag_srational`
- class `tag_srational_ptr`
- class `tag_matrix`
- class `tag_icc_profile`
- class `tag_cfa_pattern`
- class `tag_exif_date_time`
- class `tag_iptc`
- class `tag_xmp`
- class `dng_tiff_directory`
- class `dng_basic_tag_set`
- class `exif_tag_set`
- class `tiff_dng_extended_color_profile`
- class `tag_dng_noise_profile`
- class `dng_image_writer`

*Support for writing `dng_image` or `dng_negative` instances to a `dng_stream` in TIFF or DNG format.*

### 7.24.1 Detailed Description

Support for writing DNG images to files.

## 7.25 `dng_info.h` File Reference

### Classes

- class [dng\\_info](#)

*Top-level structure of DNG file with access to metadata.*

### 7.25.1 Detailed Description

Class for holding top-level information about a DNG image.

## 7.26 `dng_iptc.h` File Reference

### Classes

- class [dng\\_iptc](#)

*Class for reading and holding IPTC metadata associated with a DNG file.*

### 7.26.1 Detailed Description

Support for IPTC metadata within DNG files.

## 7.27 `dng_linearization_info.h` File Reference

### Classes

- class [dng\\_linearization\\_info](#)

*Class for managing data values related to DNG linearization.*

### 7.27.1 Detailed Description

Support for linearization table and black level tags.

## 7.28 dng\_lossless\_jpeg.h File Reference

### Classes

- class **dng\_spooler**

### Functions

- void **DecodeLosslessJPEG** ([dng\\_stream](#) &stream, **dng\_spooler** &spooler, uint32 minDecodedSize, uint32 maxDecodedSize, bool bug16)
- void **EncodeLosslessJPEG** (const uint16 \*srcData, uint32 srcRows, uint32 srcCols, uint32 srcChannels, uint32 srcBitDepth, int32 srcRowStep, int32 srcColStep, [dng\\_stream](#) &stream)

### 7.28.1 Detailed Description

Functions for encoding and decoding lossless JPEG format.

## 7.29 dng\_matrix.h File Reference

### Classes

- class **dng\_matrix**
- class **dng\_matrix\_3by3**
- class **dng\_matrix\_4by3**
- class **dng\_vector**
- class **dng\_vector\_3**

### Functions

- **dng\_matrix operator\*** (const **dng\_matrix** &A, const **dng\_matrix** &B)
- **dng\_vector operator\*** (const **dng\_matrix** &A, const **dng\_vector** &B)
- **dng\_matrix operator\*** (real64 scale, const **dng\_matrix** &A)
- **dng\_vector operator\*** (real64 scale, const **dng\_vector** &A)
- **dng\_matrix operator+** (const **dng\_matrix** &A, const **dng\_matrix** &B)
- **dng\_matrix Transpose** (const **dng\_matrix** &A)
- **dng\_matrix Invert** (const **dng\_matrix** &A)
- **dng\_matrix Invert** (const **dng\_matrix** &A, const **dng\_matrix** &hint)
- real64 **MaxEntry** (const **dng\_matrix** &A)
- real64 **MaxEntry** (const **dng\_vector** &A)
- real64 **MinEntry** (const **dng\_matrix** &A)
- real64 **MinEntry** (const **dng\_vector** &A)

### 7.29.1 Detailed Description

Matrix and vector classes, including specialized 3x3 and 4x3 versions as well as length 3 vectors.

## 7.30 `dng_memory_stream.h` File Reference

### Classes

- class [dng\\_memory\\_stream](#)  
*A `dng_stream` which can be read from or written to memory.*

### 7.30.1 Detailed Description

Stream abstraction to/from in-memory data.

## 7.31 `dng_mosaic_info.h` File Reference

### Classes

- class [dng\\_mosaic\\_info](#)  
*Support for describing color filter array patterns and manipulating mosaic sample data.*

### 7.31.1 Detailed Description

Support for descriptive information about color filter array patterns.

## 7.32 `dng_negative.h` File Reference

### Classes

- class [dng\\_noise\\_function](#)  
*Noise model for photon and sensor read noise, assuming that they are independent random variables and spatially invariant.*
- class [dng\\_noise\\_profile](#)  
*Noise profile for a negative.*

- class [dng\\_negative](#)

*Main class for holding DNG image data and associated metadata.*

### 7.32.1 Detailed Description

## 7.33 dng\_pixel\_buffer.h File Reference

### Classes

- class [dng\\_pixel\\_buffer](#)

*Holds a buffer of pixel data with "pixel geometry" metadata.*

### Defines

- #define **qDebugPixelFormat** 0
- #define **ASSERT\_PIXEL\_TYPE**(typeVal) DNG\_ASSERT (fPixelFormat == typeVal, "Pixel type access mismatch")

### Functions

- void [OptimizeOrder](#) (const void \*&sPtr, void \*&dPtr, uint32 sPixelFormat, uint32 dPixelFormat, uint32 &count0, uint32 &count1, uint32 &count2, int32 &sStep0, int32 &sStep1, int32 &sStep2, int32 &dStep0, int32 &dStep1, int32 &dStep2)  
*Compute best set of step values for a given source and destination area and stride.*
- void **OptimizeOrder** (const void \*&sPtr, uint32 sPixelFormat, uint32 &count0, uint32 &count1, uint32 &count2, int32 &sStep0, int32 &sStep1, int32 &sStep2)
- void **OptimizeOrder** (void \*&dPtr, uint32 dPixelFormat, uint32 &count0, uint32 &count1, uint32 &count2, int32 &dStep0, int32 &dStep1, int32 &dStep2)

### 7.33.1 Detailed Description

Support for holding buffers of sample data.

## 7.34 dng\_read\_image.h File Reference

### Classes

- class `dng_row_interleaved_image`
- class `dng_read_image`

#### 7.34.1 Detailed Description

Support for DNG image reading.

## 7.35 dng\_render.h File Reference

### Classes

- class `dng_function_exposure_ramp`  
*Curve for pre-exposure-compensation adjustment based on noise floor, shadows, and highlight level.*
- class `dng_function_exposure_tone`  
*Exposure compensation curve for a given compensation amount in stops using quadric for roll-off.*
- class `dng_tone_curve_acr3_default`  
*Default ACR3 tone curve.*
- class `dng_function_gamma_encode`  
*Encoding gamma curve for a given color space.*
- class `dng_render`  
*Class used to render digital negative to displayable image.*

#### 7.35.1 Detailed Description

Classes for conversion of RAW data to final image.

## 7.36 dng\_sdk\_limits.h File Reference

### Variables

- const uint32 `kMaxDNGPreviews` = 20
- const uint32 `kMaxSubIFDs` = `kMaxDNGPreviews` + 1  
*The maximum number of SubIFDs that will be parsed.*
- const uint32 `kMaxChainedIFDs` = 10  
*The maximum number of chained IFDs that will be parsed.*
- const uint32 `kMaxSamplesPerPixel` = 4  
*The maximum number of samples per pixel.*
- const uint32 `kMaxColorPlanes` = `kMaxSamplesPerPixel`  
*Maximum number of color planes.*
- const uint32 `kMaxCFAPattern` = 8  
*The maximum size of a CFA repeating pattern.*
- const uint32 `kMaxBlackPattern` = 8  
*The maximum size of a black level repeating pattern.*
- const uint32 `kMaxMaskedAreas` = 4  
*The maximum number of masked area rectangles.*
- const uint32 `kMaxImageSide` = 65000  
*The maximum image size supported (pixels per side).*
- const uint32 `kMaxMPThreads` = 8  
*Maximum number of MP threads for `dng_area_task` operations.*

### 7.36.1 Detailed Description

Collection of constants detailing maximum values used in processing in the DNG SDK.

### 7.36.2 Variable Documentation

#### 7.36.2.1 const uint32 kMaxDNGPreviews = 20

The maximum number of previews (in addition to the main IFD's thumbnail) that we support embedded in a DNG.

Referenced by `dng_image_writer::WriteDNG()`.

## Index

- ~dng\_host
  - dng\_host, [72](#)
- Allocate
  - dng\_host, [72](#)
  - dng\_memory\_allocator, [100](#)
  - dng\_memory\_data, [109](#)
- ApplyOpcodeList
  - dng\_host, [72](#)
- ApplyOrientation
  - dng\_negative, [133](#)
- Area
  - dng\_pixel\_buffer, [142](#)
- AsMemoryBlock
  - dng\_stream, [171](#)
- AutoPtr, [14](#)
  - AutoPtr, [15](#)
- BestQualityFinalHeight
  - dng\_negative, [133](#)
- BestQualityFinalWidth
  - dng\_negative, [133](#)
- BigEndian
  - dng\_stream, [172](#)
- BlackLevel
  - dng\_linearization\_info, [98](#)
- Buffer
  - dng\_memory\_block, [102](#)
  - dng\_memory\_data, [109](#)
- Buffer\_char
  - dng\_memory\_block, [102](#), [103](#)
  - dng\_memory\_data, [109](#), [110](#)
- Buffer\_int16
  - dng\_memory\_block, [103](#)
  - dng\_memory\_data, [110](#)
- Buffer\_int32
  - dng\_memory\_block, [103](#), [104](#)
  - dng\_memory\_data, [110](#), [111](#)
- Buffer\_int64
  - dng\_memory\_data, [111](#)
- Buffer\_real32
  - dng\_memory\_block, [104](#)
  - dng\_memory\_data, [111](#), [112](#)
- Buffer\_real64
  - dng\_memory\_block, [104](#), [105](#)
  - dng\_memory\_data, [112](#)
- Buffer\_uint16
  - dng\_memory\_block, [105](#)
  - dng\_memory\_data, [112](#), [113](#)
- Buffer\_uint32
  - dng\_memory\_block, [105](#), [106](#)
  - dng\_memory\_data, [113](#)
- Buffer\_uint64
  - dng\_memory\_data, [113](#), [114](#)
- Buffer\_uint8
  - dng\_memory\_block, [106](#)
  - dng\_memory\_data, [114](#)
- CalibrationIlluminant1
  - dng\_camera\_profile, [35](#)
- CalibrationIlluminant2
  - dng\_camera\_profile, [35](#)
- CalibrationTemperature1
  - dng\_camera\_profile, [35](#)
- CalibrationTemperature2
  - dng\_camera\_profile, [36](#)
- CameraToPCS
  - dng\_color\_spec, [43](#)
- CameraWhite
  - dng\_color\_spec, [43](#)
- Channels
  - dng\_color\_spec, [44](#)
- Clear
  - dng\_memory\_data, [114](#)
- ColumnBlack
  - dng\_linearization\_info, [98](#)
- ConstPixel
  - dng\_pixel\_buffer, [142](#)
- ConstPixel\_int16
  - dng\_pixel\_buffer, [143](#)
- ConstPixel\_int32
  - dng\_pixel\_buffer, [143](#)
- ConstPixel\_int8
  - dng\_pixel\_buffer, [143](#)
- ConstPixel\_real32
  - dng\_pixel\_buffer, [144](#)

- ConstPixel\_uint16
  - dng\_pixel\_buffer, 144
- ConstPixel\_uint32
  - dng\_pixel\_buffer, 145
- ConstPixel\_uint8
  - dng\_pixel\_buffer, 145
- CopyArea
  - dng\_image, 83
  - dng\_pixel\_buffer, 145, 146
- Copyright
  - dng\_camera\_profile, 36
- CopyToStream
  - dng\_memory\_stream, 116
  - dng\_stream, 172
- CurrentDateTimeAndZone
  - dng\_date\_time.h, 208
- Data
  - dng\_stream, 172
- DefaultCropArea
  - dng\_negative, 133
- DefaultScale
  - dng\_negative, 134
- DirtyPixel
  - dng\_pixel\_buffer, 146
- DirtyPixel\_int16
  - dng\_pixel\_buffer, 147
- DirtyPixel\_int32
  - dng\_pixel\_buffer, 147
- DirtyPixel\_int8
  - dng\_pixel\_buffer, 148
- DirtyPixel\_real32
  - dng\_pixel\_buffer, 148
- DirtyPixel\_uint16
  - dng\_pixel\_buffer, 148
- DirtyPixel\_uint32
  - dng\_pixel\_buffer, 149
- DirtyPixel\_uint8
  - dng\_pixel\_buffer, 149
- dng\_date\_time.h
  - dng\_date\_time\_format\_exif, 208
  - dng\_date\_time\_format\_unix\_big\_endian, 208
  - dng\_date\_time\_format\_unix\_little\_endian, 208
- dng\_date\_time\_format\_exif
  - dng\_date\_time.h, 208
- dng\_date\_time\_format\_unix\_big\_endian
  - dng\_date\_time.h, 208
- dng\_date\_time\_format\_unix\_little\_endian
  - dng\_date\_time.h, 208
- dng\_image
  - edge\_none, 83
  - edge\_repeat, 83
  - edge\_repeat\_zero\_last, 83
  - edge\_zero, 83
- dng\_1d\_concatenate, 16
  - dng\_1d\_concatenate, 16
  - dng\_1d\_concatenate, 16
- Evaluate, 17
  - EvaluateInverse, 17
- dng\_1d\_function, 18
  - Evaluate, 18
  - EvaluateInverse, 19
- dng\_1d\_function.h, 194
- dng\_1d\_identity, 19
- dng\_1d\_inverse, 20
  - Evaluate, 21
  - EvaluateInverse, 21
- dng\_1d\_table, 22
  - Initialize, 23
  - Interpolate, 23
- dng\_1d\_table.h, 195
- dng\_abort\_sniffer, 23
  - Sniff, 24
  - SniffForAbort, 24
  - StartTask, 25
  - UpdateProgress, 25
- dng\_abort\_sniffer.h, 195
- dng\_area\_task, 26
  - FindTileSize, 27
  - Finish, 27
  - MaxThreads, 27
  - MaxTileSize, 28
  - MinTaskArea, 28
  - Perform, 28
  - Process, 29
  - ProcessOnThread, 29
  - RepeatingTile1, 30
  - RepeatingTile2, 30
  - RepeatingTile3, 30

- Start, 31
- UnitCell, 31
- dng\_area\_task.h, 196
- DNG\_ASSERT
  - dng\_assertions.h, 196
- dng\_assertions.h, 196
  - DNG\_ASSERT, 196
  - DNG\_REPORT, 197
  - DNG\_REQUIRE, 197
- dng\_auto\_ptr.h, 198
- dng\_bottlenecks.h, 198
- dng\_camera\_profile, 32
  - CalibrationIlluminant1, 35
  - CalibrationIlluminant2, 35
  - CalibrationTemperature1, 35
  - CalibrationTemperature2, 36
  - Copyright, 36
  - EqualData, 36
  - HueSatMapForWhite, 36
  - IsValid, 37
  - Name, 37
  - NameIsEmbedded, 37
  - ParseExtended, 37
  - ProfileID, 37
  - SetCalibrationIlluminant1, 38
  - SetCalibrationIlluminant2, 38
  - SetColorMatrix1, 38
  - SetColorMatrix2, 38
  - SetCopyright, 39
  - SetName, 39
  - SetReductionMatrix1, 39
  - SetReductionMatrix2, 39
  - SetUniqueCameraModelRestriction, 40
  - UniqueCameraModelRestriction, 40
- dng\_camera\_profile.h, 204
- dng\_color\_space, 40
  - ICCProfile, 42
- dng\_color\_space.h, 205
- dng\_color\_spec, 42
  - CameraToPCS, 43
  - CameraWhite, 43
  - Channels, 44
  - dng\_color\_spec, 43
  - dng\_color\_spec, 43
  - NeutralToXY, 44
  - SetWhiteXY, 44
  - WhiteXY, 44
- dng\_color\_spec.h, 206
  - MapWhiteMatrix, 206
- dng\_const\_tile\_buffer, 45
  - dng\_const\_tile\_buffer, 45
  - dng\_const\_tile\_buffer, 45
- dng\_date\_time, 46
  - dng\_date\_time, 47
  - dng\_date\_time, 47
  - IsValid, 47
  - NotValid, 47
  - Parse, 47
- dng\_date\_time.h, 207
  - CurrentDateTimeAndZone, 208
  - dng\_date\_time\_format, 208
  - LocalTimeZone, 208
- dng\_date\_time\_format
  - dng\_date\_time.h, 208
- dng\_date\_time\_info, 48
- dng\_date\_time\_storage\_info, 49
  - Format, 49
  - IsValid, 50
  - Offset, 50
- dng\_dirty\_tile\_buffer, 50
  - dng\_dirty\_tile\_buffer, 51
  - dng\_dirty\_tile\_buffer, 51
- dng\_errors.h, 209
- dng\_exception, 51
  - dng\_exception, 52
  - dng\_exception, 52
  - ErrorCode, 52
- dng\_exceptions.h, 209
- dng\_exif, 52
- dng\_exif.h, 211
- dng\_fast\_module.h, 212
- dng\_file\_stream, 56
  - dng\_file\_stream, 57
  - dng\_file\_stream, 57
- dng\_file\_stream.h, 212
- dng\_filter\_task, 57
  - dng\_filter\_task, 58
  - dng\_filter\_task, 58
  - Process, 58
  - ProcessArea, 59
  - SrcArea, 59

- SrcTileSize, 60
- Start, 60
- dng\_filter\_task.h, 212
- dng\_fingerprint, 61
- dng\_fingerprint.h, 212
- dng\_flags.h, 213
- dng\_function\_exposure\_ramp, 62
  - Evaluate, 62
- dng\_function\_exposure\_tone, 63
- dng\_function\_gamma\_encode, 64
  - Evaluate, 64
- dng\_function\_GammaEncode\_1\_8, 65
  - Evaluate, 65
  - EvaluateInverse, 65
- dng\_function\_GammaEncode\_2\_2, 66
  - Evaluate, 67
  - EvaluateInverse, 67
- dng\_function\_GammaEncode\_sRGB, 68
  - Evaluate, 68
  - EvaluateInverse, 68
- dng\_globals.h, 213
- dng\_host, 69
  - ~dng\_host, 72
  - Allocate, 72
  - ApplyOpcodeList, 72
  - dng\_host, 71
  - dng\_host, 71
  - ForPreview, 72
  - IsTransientError, 73
  - Make\_dng\_exif, 73
  - Make\_dng\_ifd, 73
  - Make\_dng\_image, 73
  - Make\_dng\_negative, 74
  - Make\_dng\_opcode, 74
  - Make\_dng\_shared, 74
  - PerformAreaTask, 74
  - SetCropFactor, 75
  - SetForPreview, 75
  - SetKeepOriginalFile, 75
  - SetMaximumSize, 75
  - SetMinimumSize, 75
  - SetNeedsImage, 76
  - SetNeedsMeta, 76
  - SetPreferredSize, 76
  - SetSaveDNGVersion, 76
  - SetSaveLinearDNG, 77
  - SniffForAbort, 77
- dng\_host.h, 213
- dng\_ifd, 77
- dng\_ifd.h, 214
- dng\_image, 81
  - CopyArea, 83
  - edge\_option, 83
  - EqualArea, 84
  - Get, 84
  - PixelRange, 85
  - PixelSize, 85
  - PixelType, 85
  - Put, 85
  - Rotate, 86
  - SetPixelType, 86
  - Trim, 86
- dng\_image.h, 214
- dng\_image\_writer, 87
  - WriteDNG, 88
  - WriteTIFF, 89
  - WriteTIFFWithProfile, 90
- dng\_image\_writer.h, 215
- dng\_info, 91
  - IsValidDNG, 92
  - Parse, 92
- dng\_info.h, 216
- dng\_iptc, 93
  - IsEmpty, 95
  - NotEmpty, 95
  - Parse, 95
  - Spool, 96
- dng\_iptc.h, 216
- dng\_linearization\_info, 96
  - BlackLevel, 98
  - ColumnBlack, 98
  - fActiveArea, 99
  - fLinearizationTable, 99
  - fMaskedArea, 100
  - Linearize, 98
  - RowBlack, 99
- dng\_linearization\_info.h, 216
- dng\_lossless\_jpeg.h, 217
- dng\_matrix.h, 217
- dng\_memory\_allocator, 100
  - Allocate, 100
- dng\_memory\_block, 101

- Buffer, 102
- Buffer\_char, 102, 103
- Buffer\_int16, 103
- Buffer\_int32, 103, 104
- Buffer\_real32, 104
- Buffer\_real64, 104, 105
- Buffer\_uint16, 105
- Buffer\_uint32, 105, 106
- Buffer\_uint8, 106
- LogicalSize, 106
- dng\_memory\_data, 107
  - Allocate, 109
  - Buffer, 109
  - Buffer\_char, 109, 110
  - Buffer\_int16, 110
  - Buffer\_int32, 110, 111
  - Buffer\_int64, 111
  - Buffer\_real32, 111, 112
  - Buffer\_real64, 112
  - Buffer\_uint16, 112, 113
  - Buffer\_uint32, 113
  - Buffer\_uint64, 113, 114
  - Buffer\_uint8, 114
  - Clear, 114
  - dng\_memory\_data, 108
  - dng\_memory\_data, 108
- dng\_memory\_stream, 115
  - CopyToStream, 116
  - dng\_memory\_stream, 116
  - dng\_memory\_stream, 116
- dng\_memory\_stream.h, 218
- dng\_mosaic\_info, 116
  - DownScale, 118
  - DstSize, 118
  - fBayerGreenSplit, 121
  - fCFALayout, 121
  - FullScale, 119
  - Interpolate, 119
  - InterpolateFast, 119
  - InterpolateGeneric, 120
  - IsColorFilterArray, 120
  - SetFourColorBayer, 121
- dng\_mosaic\_info.h, 218
- dng\_negative, 122
  - ApplyOrientation, 133
  - BestQualityFinalHeight, 133
  - BestQualityFinalWidth, 133
  - DefaultCropArea, 133
  - DefaultScale, 134
  - SetCameraCalibration1, 134
  - SetCameraCalibration2, 134
- dng\_negative.h, 218
- dng\_noise\_function, 135
  - Evaluate, 135
- dng\_noise\_profile, 136
- dng\_opcode\_FixVignetteRadial, 137
- dng\_opcode\_WarpFisheye, 138
- dng\_opcode\_WarpRectilinear, 139
- dng\_pixel\_buffer, 140
  - Area, 142
  - ConstPixel, 142
  - ConstPixel\_int16, 143
  - ConstPixel\_int32, 143
  - ConstPixel\_int8, 143
  - ConstPixel\_real32, 144
  - ConstPixel\_uint16, 144
  - ConstPixel\_uint32, 145
  - ConstPixel\_uint8, 145
  - CopyArea, 145, 146
  - DirtyPixel, 146
  - DirtyPixel\_int16, 147
  - DirtyPixel\_int32, 147
  - DirtyPixel\_int8, 148
  - DirtyPixel\_real32, 148
  - DirtyPixel\_uint16, 148
  - DirtyPixel\_uint32, 149
  - DirtyPixel\_uint8, 149
  - EqualArea, 150
  - FlipH, 150
  - FlipV, 150
  - FlipZ, 150
  - MaximumDifference, 151
  - PixelRange, 151
  - Planes, 151
  - PlaneStep, 152
  - RepeatArea, 152
  - RepeatPhase, 152
  - RowStep, 152
  - SetConstant, 153
  - SetConstant\_int16, 153
  - SetConstant\_real32, 153
  - SetConstant\_uint16, 154

- SetConstant\_uint32, 154
- SetConstant\_uint8, 154
- SetZero, 155
- ShiftRight, 155
- dng\_pixel\_buffer.h, 219
- dng\_read\_image.h, 220
- dng\_render, 156
  - dng\_render, 157
  - dng\_render, 157
  - Exposure, 157
  - FinalPixelFormat, 157
  - FinalSpace, 157
  - MaximumSize, 158
  - Render, 158
  - SetExposure, 158
  - SetFinalPixelFormat, 158
  - SetFinalSpace, 159
  - SetMaximumSize, 159
  - SetShadows, 159
  - SetToneCurve, 159
  - SetWhiteXY, 160
  - Shadows, 160
  - ToneCurve, 160
  - WhiteXY, 160
- dng\_render.h, 220
- DNG\_REPORT
  - dng\_assertions.h, 197
- DNG\_REQUIRE
  - dng\_assertions.h, 197
- dng\_sdk\_limits.h, 221
- kMaxDNGPreviews, 221
- dng\_simple\_image, 161
- dng\_sniffer\_task, 162
  - dng\_sniffer\_task, 163
  - dng\_sniffer\_task, 163
  - Sniff, 163
  - UpdateProgress, 163
- dng\_space\_AdobeRGB, 164
- dng\_space\_ColorMatch, 165
- dng\_space\_GrayGamma18, 165
- dng\_space\_GrayGamma22, 166
- dng\_space\_ProPhoto, 167
- dng\_space\_sRGB, 168
- dng\_stream, 169
  - AsMemoryBlock, 171
  - BigEndian, 172
  - CopyToStream, 172
  - Data, 172
  - dng\_stream, 171
  - dng\_stream, 171
  - DuplicateStream, 172
  - Get, 173
  - Get\_CString, 173
  - Get\_int16, 173
  - Get\_int32, 174
  - Get\_int64, 174
  - Get\_int8, 175
  - Get\_real32, 175
  - Get\_real64, 175
  - Get\_uint16, 176
  - Get\_uint32, 176
  - Get\_uint64, 177
  - Get\_uint8, 177
  - Get\_UString, 178
  - Length, 178
  - LittleEndian, 178
  - OffsetInOriginalFile, 179
  - Position, 179
  - PositionInOriginalFile, 179
  - Put, 179
  - Put\_int16, 180
  - Put\_int32, 180
  - Put\_int64, 180
  - Put\_int8, 181
  - Put\_real32, 181
  - Put\_real64, 181
  - Put\_uint16, 181
  - Put\_uint32, 182
  - Put\_uint64, 182
  - Put\_uint8, 182
  - PutZeros, 183
  - SetBigEndian, 183
  - SetLength, 183
  - SetLittleEndian, 183
  - SetSniffer, 184
  - SetSwapBytes, 184
  - Skip, 184
  - Sniffer, 184
  - SwapBytes, 185
  - TagValue\_int32, 185
  - TagValue\_real64, 185
  - TagValue\_srational, 186

- TagValue\_uint32, 186
- TagValue\_urational, 187
- dng\_tile\_buffer, 188
  - dng\_tile\_buffer, 188
  - dng\_tile\_buffer, 188
- dng\_time\_zone, 189
- dng\_tone\_curve\_acr3\_default, 190
- dng\_vignette\_radial\_params, 190
- dng\_warp\_params, 191
- dng\_warp\_params\_fisheye, 192
- dng\_warp\_params\_rectilinear, 193
- DownScale
  - dng\_mosaic\_info, 118
- DstSize
  - dng\_mosaic\_info, 118
- DuplicateStream
  - dng\_stream, 172
- edge\_none
  - dng\_image, 83
- edge\_repeat
  - dng\_image, 83
- edge\_repeat\_zero\_last
  - dng\_image, 83
- edge\_zero
  - dng\_image, 83
- edge\_option
  - dng\_image, 83
- EqualArea
  - dng\_image, 84
  - dng\_pixel\_buffer, 150
- EqualData
  - dng\_camera\_profile, 36
- ErrorCode
  - dng\_exception, 52
- Evaluate
  - dng\_1d\_concatenate, 17
  - dng\_1d\_function, 18
  - dng\_1d\_inverse, 21
  - dng\_function\_exposure\_ramp, 62
  - dng\_function\_gamma\_encode, 64
  - dng\_function\_GammaEncode\_1\_8, 65
  - dng\_function\_GammaEncode\_2\_2, 67
- dng\_function\_GammaEncode\_-sRGB, 68
- dng\_noise\_function, 135
- EvaluateInverse
  - dng\_1d\_concatenate, 17
  - dng\_1d\_function, 19
  - dng\_1d\_inverse, 21
  - dng\_function\_GammaEncode\_1\_8, 65
  - dng\_function\_GammaEncode\_2\_2, 67
  - dng\_function\_GammaEncode\_-sRGB, 68
- Exposure
  - dng\_render, 157
- fActiveArea
  - dng\_linearization\_info, 99
- fBayerGreenSplit
  - dng\_mosaic\_info, 121
- fCFALayout
  - dng\_mosaic\_info, 121
- FinalPixelType
  - dng\_render, 157
- FinalSpace
  - dng\_render, 157
- FindTileSize
  - dng\_area\_task, 27
- Finish
  - dng\_area\_task, 27
- fLinearizationTable
  - dng\_linearization\_info, 99
- FlipH
  - dng\_pixel\_buffer, 150
- FlipV
  - dng\_pixel\_buffer, 150
- FlipZ
  - dng\_pixel\_buffer, 150
- fMaskedArea
  - dng\_linearization\_info, 100
- Format
  - dng\_date\_time\_storage\_info, 49
- ForPreview
  - dng\_host, 72
- FullScale
  - dng\_mosaic\_info, 119

- Get
    - dng\_image, 84
    - dng\_stream, 173
  - Get\_CString
    - dng\_stream, 173
  - Get\_int16
    - dng\_stream, 173
  - Get\_int32
    - dng\_stream, 174
  - Get\_int64
    - dng\_stream, 174
  - Get\_int8
    - dng\_stream, 175
  - Get\_real32
    - dng\_stream, 175
  - Get\_real64
    - dng\_stream, 175
  - Get\_uint16
    - dng\_stream, 176
  - Get\_uint32
    - dng\_stream, 176
  - Get\_uint64
    - dng\_stream, 177
  - Get\_uint8
    - dng\_stream, 177
  - Get\_UString
    - dng\_stream, 178
- HueSatMapForWhite
  - dng\_camera\_profile, 36
- ICCProfile
  - dng\_color\_space, 42
- Initialize
  - dng\_1d\_table, 23
- Interpolate
  - dng\_1d\_table, 23
  - dng\_mosaic\_info, 119
- InterpolateFast
  - dng\_mosaic\_info, 119
- InterpolateGeneric
  - dng\_mosaic\_info, 120
- IsColorFilterArray
  - dng\_mosaic\_info, 120
- IsEmpty
  - dng\_iptc, 95
- IsTransientError
  - dng\_host, 73
- IsValid
  - dng\_camera\_profile, 37
  - dng\_date\_time, 47
  - dng\_date\_time\_storage\_info, 50
- IsValidDNG
  - dng\_info, 92
- kMaxDNGPreviews
  - dng\_sdk\_limits.h, 221
- Length
  - dng\_stream, 178
- Linearize
  - dng\_linearization\_info, 98
- LittleEndian
  - dng\_stream, 178
- LocalTimeZone
  - dng\_date\_time.h, 208
- LogicalSize
  - dng\_memory\_block, 106
- Make\_dng\_exif
  - dng\_host, 73
- Make\_dng\_ifd
  - dng\_host, 73
- Make\_dng\_image
  - dng\_host, 73
- Make\_dng\_negative
  - dng\_host, 74
- Make\_dng\_opcode
  - dng\_host, 74
- Make\_dng\_shared
  - dng\_host, 74
- MapWhiteMatrix
  - dng\_color\_spec.h, 206
- MaximumDifference
  - dng\_pixel\_buffer, 151
- MaximumSize
  - dng\_render, 158
- MaxThreads
  - dng\_area\_task, 27
- MaxTileSize
  - dng\_area\_task, 28
- MinTaskArea

- dng\_area\_task, 28
  - Name
    - dng\_camera\_profile, 37
  - NameIsEmbedded
    - dng\_camera\_profile, 37
  - NeutralToXY
    - dng\_color\_spec, 44
  - NotEmpty
    - dng\_iptc, 95
  - NotValid
    - dng\_date\_time, 47
  - Offset
    - dng\_date\_time\_storage\_info, 50
  - OffsetInOriginalFile
    - dng\_stream, 179
  - Parse
    - dng\_date\_time, 47
    - dng\_info, 92
    - dng\_iptc, 95
  - ParseExtended
    - dng\_camera\_profile, 37
  - Perform
    - dng\_area\_task, 28
  - PerformAreaTask
    - dng\_host, 74
  - PixelRange
    - dng\_image, 85
    - dng\_pixel\_buffer, 151
  - PixelSize
    - dng\_image, 85
  - PixelType
    - dng\_image, 85
  - Planes
    - dng\_pixel\_buffer, 151
  - PlaneStep
    - dng\_pixel\_buffer, 152
  - Position
    - dng\_stream, 179
  - PositionInOriginalFile
    - dng\_stream, 179
  - Process
    - dng\_area\_task, 29
    - dng\_filter\_task, 58
  - ProcessArea
    - dng\_filter\_task, 59
  - ProcessOnThread
    - dng\_area\_task, 29
  - ProfileID
    - dng\_camera\_profile, 37
  - Put
    - dng\_image, 85
    - dng\_stream, 179
  - Put\_int16
    - dng\_stream, 180
  - Put\_int32
    - dng\_stream, 180
  - Put\_int64
    - dng\_stream, 180
  - Put\_int8
    - dng\_stream, 181
  - Put\_real32
    - dng\_stream, 181
  - Put\_real64
    - dng\_stream, 181
  - Put\_uint16
    - dng\_stream, 181
  - Put\_uint32
    - dng\_stream, 182
  - Put\_uint64
    - dng\_stream, 182
  - Put\_uint8
    - dng\_stream, 182
  - PutZeros
    - dng\_stream, 183
- Render
    - dng\_render, 158
  - RepeatArea
    - dng\_pixel\_buffer, 152
  - RepeatingTile1
    - dng\_area\_task, 30
  - RepeatingTile2
    - dng\_area\_task, 30
  - RepeatingTile3
    - dng\_area\_task, 30
  - RepeatPhase
    - dng\_pixel\_buffer, 152
  - Rotate
    - dng\_image, 86

- RowBlack
  - dng\_linearization\_info, 99
- RowStep
  - dng\_pixel\_buffer, 152
- SetBigEndian
  - dng\_stream, 183
- SetCalibrationIlluminant1
  - dng\_camera\_profile, 38
- SetCalibrationIlluminant2
  - dng\_camera\_profile, 38
- SetCameraCalibration1
  - dng\_negative, 134
- SetCameraCalibration2
  - dng\_negative, 134
- SetColorMatrix1
  - dng\_camera\_profile, 38
- SetColorMatrix2
  - dng\_camera\_profile, 38
- SetConstant
  - dng\_pixel\_buffer, 153
- SetConstant\_int16
  - dng\_pixel\_buffer, 153
- SetConstant\_real32
  - dng\_pixel\_buffer, 153
- SetConstant\_uint16
  - dng\_pixel\_buffer, 154
- SetConstant\_uint32
  - dng\_pixel\_buffer, 154
- SetConstant\_uint8
  - dng\_pixel\_buffer, 154
- SetCopyright
  - dng\_camera\_profile, 39
- SetCropFactor
  - dng\_host, 75
- SetExposure
  - dng\_render, 158
- SetFinalPixelType
  - dng\_render, 158
- SetFinalSpace
  - dng\_render, 159
- SetForPreview
  - dng\_host, 75
- SetFourColorBayer
  - dng\_mosaic\_info, 121
- SetKeepOriginalFile
  - dng\_host, 75
- SetLength
  - dng\_stream, 183
- SetLittleEndian
  - dng\_stream, 183
- SetMaximumSize
  - dng\_host, 75
  - dng\_render, 159
- SetMinimumSize
  - dng\_host, 75
- SetName
  - dng\_camera\_profile, 39
- SetNeedsImage
  - dng\_host, 76
- SetNeedsMeta
  - dng\_host, 76
- SetPixelType
  - dng\_image, 86
- SetPreferredSize
  - dng\_host, 76
- SetReductionMatrix1
  - dng\_camera\_profile, 39
- SetReductionMatrix2
  - dng\_camera\_profile, 39
- SetSaveDNGVersion
  - dng\_host, 76
- SetSaveLinearDNG
  - dng\_host, 77
- SetShadows
  - dng\_render, 159
- SetSniffer
  - dng\_stream, 184
- SetSwapBytes
  - dng\_stream, 184
- SetToneCurve
  - dng\_render, 159
- SetUniqueCameraModelRestriction
  - dng\_camera\_profile, 40
- SetWhiteXY
  - dng\_color\_spec, 44
  - dng\_render, 160
- SetZero
  - dng\_pixel\_buffer, 155
- Shadows
  - dng\_render, 160
- ShiftRight

- dng\_pixel\_buffer, 155
- Skip
  - dng\_stream, 184
- Sniff
  - dng\_abort\_sniffer, 24
  - dng\_sniffer\_task, 163
- Sniffer
  - dng\_stream, 184
- SniffForAbort
  - dng\_abort\_sniffer, 24
  - dng\_host, 77
- Spool
  - dng\_iptc, 96
- SrcArea
  - dng\_filter\_task, 59
- SrcTileSize
  - dng\_filter\_task, 60
- Start
  - dng\_area\_task, 31
  - dng\_filter\_task, 60
- StartTask
  - dng\_abort\_sniffer, 25
- SwapBytes
  - dng\_stream, 185
- TagValue\_int32
  - dng\_stream, 185
- TagValue\_real64
  - dng\_stream, 185
- TagValue\_srational
  - dng\_stream, 186
- TagValue\_uint32
  - dng\_stream, 186
- TagValue\_urational
  - dng\_stream, 187
- ToneCurve
  - dng\_render, 160
- Trim
  - dng\_image, 86
- UniqueCameraModelRestriction
  - dng\_camera\_profile, 40
- UnitCell
  - dng\_area\_task, 31
- UpdateProgress
  - dng\_abort\_sniffer, 25
  - dng\_sniffer\_task, 163
- WhiteXY
  - dng\_color\_spec, 44
  - dng\_render, 160
- WriteDNG
  - dng\_image\_writer, 88
- WriteTIFF
  - dng\_image\_writer, 89
- WriteTIFFWithProfile
  - dng\_image\_writer, 90